Design and Implementation of Improved Decoding Algorithms for LDPC Convolutional Codes

Entwurf und Implementierung von verbesserten Decodieralgorithmen für LDPC-Faltungscodes

Master-Thesis von Sakthivel Velumani aus Salem, Indien

Tag der Einreichung: 14.01.2019

1. Gutachten: Janik Frenzel, M.Sc.
2. Gutachten: Bastian Alt, M.Sc.
3. Gutachten: Prof. Dr. techn. Heinz Köppl

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Elektrotechnik und Informationstechnik
Bioinspired Communication Systems Lab

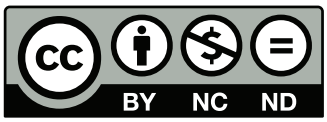
Design and Implementation of Improved Decoding Algorithms for LDPC Convolutional Codes
Entwurf und Implementierung von verbesserten Decodieralgorithmen für LDPC-Faltungscodes

Vorgelegte Master-Thesis von Sakthivel Velumani aus Salem, Indien

Matrikelnummer: 2806422

1. Gutachten: Janik Frenzel, M.Sc.
2. Gutachten: Bastian Alt, M.Sc.
3. Gutachten: Prof. Dr. techn. Heinz Köppl

Tag der Einreichung: 14.01.2019



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Sakthivel Velumani, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zuhaben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

English translation for information purposes only:

Thesis Statement pursuant to § 22 paragraph 7 and § 23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Sakthivel Velumani have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB identical in content.

Datum / Date: Unterschrift / Signature:

_____ _____

Acknowledgment

This work represents the results of the six-month Master's thesis project carried out at Intel Deutschland GmbH, Nuremberg in cooperation with Bioinspired Communication Systems Lab at Technische Universität Darmstadt.

First of all, I would like to thank my supervisor Janik Frenzel, M.Sc. and his supervisor Dr.-Ing. Stefan Müller-Weinfurtner for having faith in me and give me this wonderful opportunity. I would like to also thank my supervisor at the university Bastian Alt, M.Sc. and the head of the group Prof. Dr. techn. Heinz Köppl for accepting my thesis work done at Intel Deutschland GmbH.

Especially, I would like to thank Janik Frenzel for introducing me to the field of LDPC codes with a practical and industrial perspective. The several discussions we had during this project certainly improved my insights into the topic and helped to overcome the ambiguities in my understanding. I greatly appreciate his patience and efforts to answer my questions as well as to review this work.

Special thanks to my family and friends in India for providing moral support. Without them, this work would not have been possible.

Abstract

A windowed decoder in its basic form converges rather slowly and has a large performance gap to a full-block decoder. In this work, we propose two techniques to improve the performance of windowed decoders for Low-Density Parity-Check Convolutional Codes (LDPC-CCs). The first technique: the LRL decoder, focuses on the movement direction of the window in which the window moves forward and backward across the Parity-Check Matrix (PCM). The second technique: the IPSC, focuses on the convergence criterion for the windows where the criterion is dependent on window size. We chose the LDPC-CCs specified in the standard IEEE 1901 Broadband Power Line (BPL) to evaluate our techniques. We found that a proper end-termination for the BPL's LDPC-CCs is infeasible. We show that although the termination procedure mentioned in the standard fails to reduces the Check Node (CN) degree, the known termination bits at the decoder effectively reduce the CN degree. Simulation results show that compared to the sliding-window decoder, the LRL decoder has a decoding performance gain of about 1.6 dB while simultaneously reducing the decoding complexity by up to 40%. On the other hand, the IPSC technique proves to reduce the decoding complexity by up to 34% depending on the window sizes.

Contents

List of Figures

List of Tables

Acronyms

APP	A-Posteriori Probability
AWGN	Additive White Gaussian Noise
BD	Base Decoder
BER	Bit-Error Rate
BLER	Block-Error Rate
BP	Belief Propagation
BPL	Broadband Power Line
BPSK	Binary Phase Shift Keying
C2V	check-node–to–variable-node
CB	Code Block
CN	Check Node
FIR	Finite Impulse Response
GM	Generator Matrix
GSM	Global System for Mobile communications
IIR	Infinite Impulse Response
IPSC	Improved Partial-Syndrome-Check
LDPC	Low-Density Parity-Check
LDPC-BC	Low-Density Parity-Check Block Code
LDPC-CC	Low-Density Parity-Check Convolutional Code
LLR	Log-Likelihood Ratio
LRL	Left-to-Right-to-Left
LSB	Least Significant Bit
LTE	Long Term Evloution
MAP	Maximum-A-Posteriori
ML	Maximum-Likelihood
MPA	Message-Passing Algorithm
MSA	Min-Sum Algorithm
NR	New Radio
OSI	Open System Interconnection

PCM	Parity-Check Matrix
pdf	Probability Density Function
PHY	Physical
PSC	Partial-Syndrome-Check
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
SC-LDPC	Spatially-Coupled LDPC
SNR	Signal-to-Noise Ratio
SPA	Sum-Product algorithm
TNEU	Total Number of Edge Updates
UMTS	Universal Mobile Telecommunications System
V2C	variable-node–to–check-node
VN	Variable Node
WiMAX	Worldwide Interoperability for Microwave Access

List of Symbols

I	Maximum number of iterations allowed within a window
L	Coupling Length
P_L	Probability of block error
P_b	Probability of bit error
R_∞	Asymptotic Code Rate
R	Actual Code Rate
W	Window Size
η	Total Number of Edge Updates (TNEU)
κ	Decoder's output bit rate
$\mathbf{G}$	Generator Matrix
$\mathbf{H}$	Parity-Check Matrix
$\mathbf{b}$	Vector of parity bits
$\mathbf{d}$	Vector of source bits
$\mathbf{m}$	Vector of source encoded (information) bits
$\mathbf{s}$	Syndrome
$\mathbf{u}$	Vector of modulated symbols
$\mathbf{v}$	Vector of received symbols
$\mathbf{x}$	Vector of codeword bits
$\mathbf{y}$	Vector of received codeword bits
$\mathbf{z}$	Vector of noise samples
ψ	Time taken to perform one CN Processing
ρ	Window position in a window decoder
Pr	Probability
$\widehat{\mathbf{d}}$	Vector of estimated source bits
$\widehat{\mathbf{m}}$	Vector of estimated information bits
$\widehat{\mathbf{x}}$	Vector of estimated codeword bits
ζ	Signal-to-Noise Ratio (SNR) in dB
k	Number of information bits in a Code Block
l_c	Constraint Length

m_s	Memory of a LDPC-CCs
n_m	Number of information bits in the codeword
n_z	Number of Zero-tail bits in the codeword
n	Number of bits in a Code Block
o	Modulation Order
p	Probability Density Function
q	Variable Node degree
r	Check Node degree

1 Introduction

Error-correcting codes are a vital part of any reliable communication system. Since the first use of error-correcting codes, they have evolved to a great extent. Convolutional codes are used in Global System for Mobile communications (GSM) (2G) [1] which is the first standardized digital cellular network. After the introduction of Turbo codes, they are being used in Universal Mobile Telecommunications System (UMTS) (3G) [2] and Long Term Evloution (LTE) (4G) [3]. The low decoding complexities and high performance of Low-Density Parity-Check (LDPC) codes made them better candidates for the next generation mobile cellular technology: the New Radio (NR) (5G) [4]. NR uses a set of Low-Density Parity-Check Block Codes (LDPC-BCs) for user data transmissions and Polar codes for transferring all other control information.

Terminated LDPC-CCs with large codeword lengths are proven to be better than LDPC-BCs [5]. Alongside the code's performance, the LDPC-CCs have an advantage in their decodability. Its convolutional structure allows for windowed decoding which requires fewer resources than a full-block decoder. However, windowed decoding always has a trade-off between decoding latency and decoding performance. LDPC-CCs are used in applications like BPL (IEEE 1901) [6] and Worldwide Interoperability for Microwave Access (WiMAX) (IEEE 802.16) [7] standards. But due to the better performance and increasing popularity of LDPC-CC, their applications are more likely to be increased in the future. Figure 1.1 shows the evolution of mobile cellular technologies and error-correcting codes used.

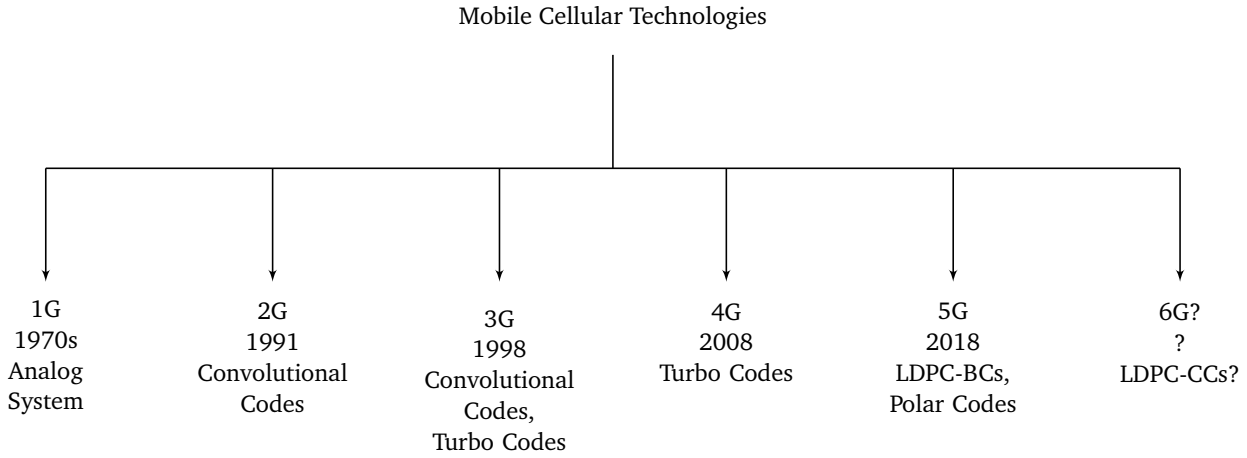


Figure 1.1: Evolution of mobile telephone technologies and type of error-correcting codes used.

In this thesis, we propose techniques to efficiently decode LDPC-CCs. The LRL decoder moves the decoding window forward and backward within the PCM. The Improved Partial-Syndrome-Check (IPSC) is an improvement over Partial-Syndrome-Check (PSC) proposed in [8]. We also investigate the termination problem in the LDPC-CCs used in the standard IEEE 1901.

The organization of this thesis is as follows. Chapter 2 contains the theory of channel coding and channel codes with an emphasis on LDPC-CCs. We also discuss the system model used for our simulations. In Chapter 3, we discuss the termination problem in IEEE 1901's LDPC-CCs. We then discuss the proposed decoding techniques and their evaluation in Chapters 4 and 5, respectively. Chapter 6 contains details about implementation of the decoder. Finally in Chapter 7, we conclude our findings of this work and discuss about possible future works.

2 Background

In this chapter, we provide an overview of channel coding and some selected channel codes. We start by discussing the need for channel coding. We then describe linear block codes, LDPC codes and LDPC-CCs which are the main focus of this thesis. Then we go on to discuss the Belief Propagation (BP) and the sliding-window technique used in decoders for LDPC-CCs. We finish the section describing our system model.

2.1 Introduction to Channel Coding

The term *coding* is generally associated with the mapping of information to a set of symbols or numbers [9]. Source coding aims to remove the redundancy in the information whereas channel coding aims to make the information immune to random distortion. A model of a digital communication system is shown in Figure 2.1. Let us consider that the *source* block produces a sequence of information bits given by the vector $\mathbf{d}$. These bits might stream from any digital information source such as multimedia files, text documents, etc. These vectors of bits are encoded in the *source-encoder* block to produce reduced set of information bit vectors $\mathbf{m}$ called source codewords. The reduction usually means that the length of $\mathbf{m}$ is at most the length of $\mathbf{d}$. The mapping of information bits to a set of reduced information bits allows unique reconstruction of the information bits at the receiver. The source encoder is chosen depending on the type of the information source [10].

The next block in the digital communication model is the *channel encoder*. Whereas the source encoder compresses the information bit vectors, the channel encoder expands them by adding redundant bits in a structured manner. This structured redundancy makes the transmitted information bits less susceptible to distortions such as interference in the channel noise. A *channel* is a medium through which the information is transferred from transmitter to receiver. A *code* is a set of rules that defines the encoding principle of the encoder. The type of code is chosen depending on the channel and the application requirements. In general, the source encoder or decoder is placed at higher layers of the Open System Interconnection (OSI) model, while channel-coding blocks are placed at the Physical (PHY) layer. The outputs of the channel encoder are called channel codewords. The codeword vectors $\mathbf{x}$ are then modulated in the *modulator* block where the bits are transformed into symbol vectors $\mathbf{u}$. The symbol vectors are then transmitted as analog signals through the channel. Due to the addition of interference and noise, the channel output is in general not the same as the channel input: $\mathbf{v} \neq \mathbf{u}$. The *demodulator* converts the received symbol vectors $\mathbf{v}$ into vectors of bits $\mathbf{y}$ which corresponds to the vector of encoded bits $\mathbf{x}$. The *channel decoder* uses the redundancy in the received codeword to deduce an estimate $\widehat{\mathbf{m}}$ of the source codeword. The source decoder then deduces an estimate $\widehat{\mathbf{d}}$ of the information bit vector from $\widehat{\mathbf{m}}$.

The addition of redundant bits by the channel encoder enables the mapping between a set of information words and a set of all possible *receive words*. Let us assume the length of an information word $\mathbf{m}$ to be k bits and the length of a codeword $\mathbf{x}$ to be n bits such that $n > k$. Thus, the information word set has 2^k words and the receive word set has 2^n words. The codeword set of size 2^k is a subset of the receive word set. The mapping between different set sizes allows us to detect if the received word is in the codeword set. The information words and codewords contain elements from the binary set $\mathbb{F}_2 = \{0, 1\}$. $\mathbb{F}_2$ or GF(2) is called a finite field of order 2. Hence, all arithmetic operations with information bits and codewords are performed modulo 2.

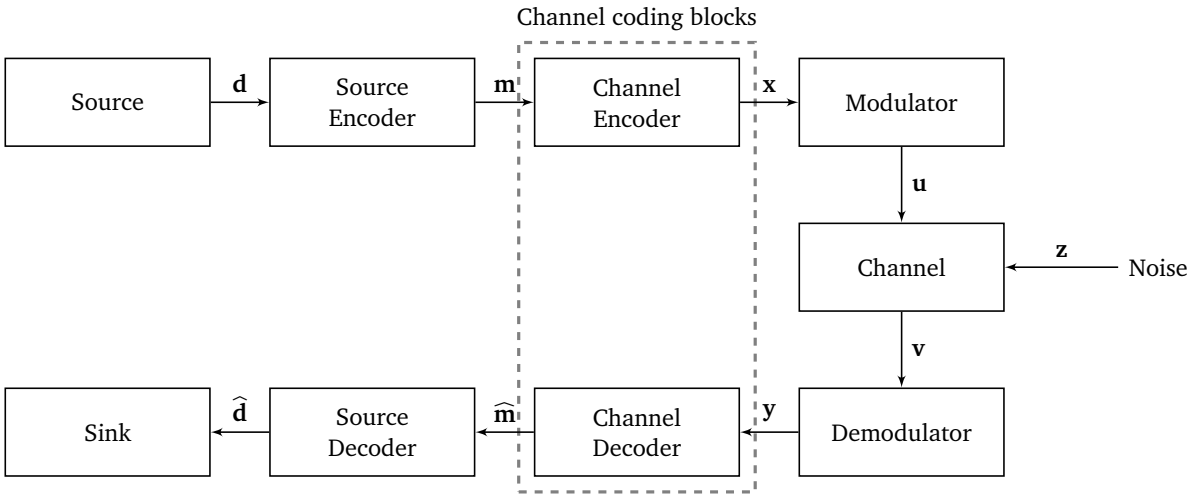


Figure 2.1: Block diagram of a digital communication system.

2.2 Channel Codes

There are different types of channel codes. The choice of one depends on the application requirements, type of channel medium and resource availability. In this thesis, we focus on LDPC-CCs, a special class of LDPC codes.

2.2.1 Linear Block Codes

Linear block codes are codes in which a codeword is formed by a linear combination of two or more base vectors that span the codeword space [10]. Hence, the base vectors are also codewords. As a result, a linear combination of any two or more codewords forms another codeword. The codeword space of 2^k vectors is a subspace of the space of all 2^n vectors. An (n,k) linear block code maps k message bits to n codeword bits. The remaining $n-k$ redundant bits are called parity bits and they are determined by an encoding rule. Linear block codes are classified into two categories: *systematic* and *non-systematic*. Systematic codes have all their message bits transmitted in an unaltered manner whereas the non-systematic codes do not have such formation. Without loss of generality, we assume that a codeword of a systematic linear block code has the following structure:

$$\mathbf{x}^T = \begin{bmatrix} \mathbf{m}^T & \mathbf{b}^T \end{bmatrix} \tag{2.1}$$

where $\mathbf{x} \in \mathbb{F}_2^{n\times 1}$ is the codeword vector, $\mathbf{m} \in \mathbb{F}_2^{k\times 1}$ and $\mathbf{b} \in \mathbb{F}_2^{(n-k)\times 1}$ denote message and parity vectors, respectively. The code rate is given by

$$R = \frac{k}{n}. \tag{2.2}$$

Codewords of linear block codes are expressed as

$$\mathbf{x} = \mathbf{G} \odot \mathbf{m} \tag{2.3}$$

where $\mathbf{G} \in \mathbb{F}_2^{n \times k}$ is the Generator Matrix (GM) and $\odot$ represents multiplication modulo 2. A parity check is described by the expression

$$\mathbf{s} = \mathbf{H} \odot \mathbf{x} \tag{2.4}$$

where $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ is called the PCM and $\mathbf{s} \in \mathbb{F}_2^{(n-k) \times 1}$ is called the syndrome. Each row of the PCM represents a parity-check equation. Only when $\mathbf{s} = \mathbf{0}$, the parity checks are fulfilled. The relation between PCM and GM is given by $\mathbf{H} \odot \mathbf{G} = \mathbf{0}$. With either GM or PCM given, the other one is not unique. For example, if the PCM of a $(7, 4)$ hamming code is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{2.5}$$

then the GM can be formed by combining any 3 rows of null($\mathbf{H}$), i.e., the right null space of $\mathbf{H}$.

$$\mathbf{G}^T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.6}$$

Figure 2.2 shows a simple comparison of Bit-Error Rate (BER) between uncoded transmission, transmission using a $(7, 4)$ hamming code and transmission using an LDPC with about the same rate as the hamming code.

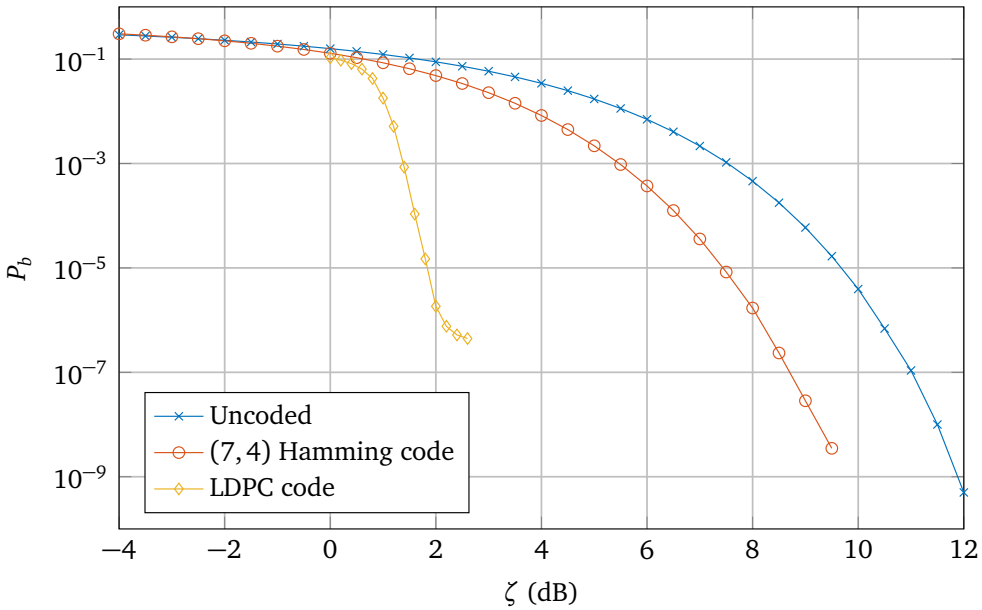


Figure 2.2: Probability of bit error for uncoded, Hamming coded and LDPC coded transmissions. The LDPC codes has a code rate of 0.5 and codeword length of 5000.

The PCM can be represented by a bipartite graph called Tanner graph [11]. The Tanner graph has two sets of nodes: VNs represent columns and CNs represent rows of the PCM. Each non-zero entry in the

PCM is represented by an edge between the respective VN and CN. The *degree* of a node is the number of edges connected to it. The Tanner graph of the example PCM in (2.5) is shown in Figure 2.3.

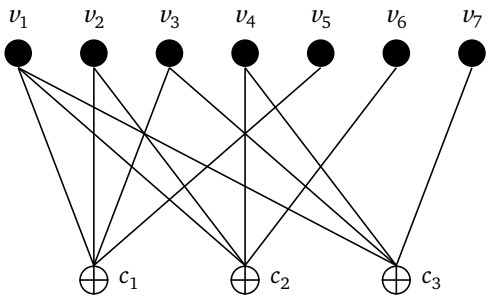


Figure 2.3: Tanner graph of the code from (2.5). The dark shaded circles represents VNs and the crossed circles CNs. All CNs have degree 3 whereas the VN degrees vary between 1 and 3.

2.2.2 Low-Density Parity-Check Block Codes

LDPC-BCs are a class of linear block codes which were introduced by Robert Gallager in 1963 [12]. As the name specifies, they are defined by a sparse PCM containing mostly 0's and relatively few 1's. The sparsity of the PCM or its Tanner graph is a key property that allows for the algorithmic efficiency of decoding LDPC-BCs. These codes are divided into two types: regular and irregular codes.

In a regular (n, q, r) code, all VNs have degree q and all CNs have degree r.

2.2.3 Convolutional Codes

Convolutional codes in general, are codes in which the parity bits are generated by convolving information bits or information and parity bits. The polynomial coefficients for the convolution is given by the generator polynomial. This generator polynomial is also the taps of a Finite Impulse Response (FIR) filter in case of non-recursive codes and an Infinite Impulse Response (IIR) filter in case of recursive codes. An example of parity-bit generation in a non-recursive systematic convolutional code is shown in Figure 2.4.

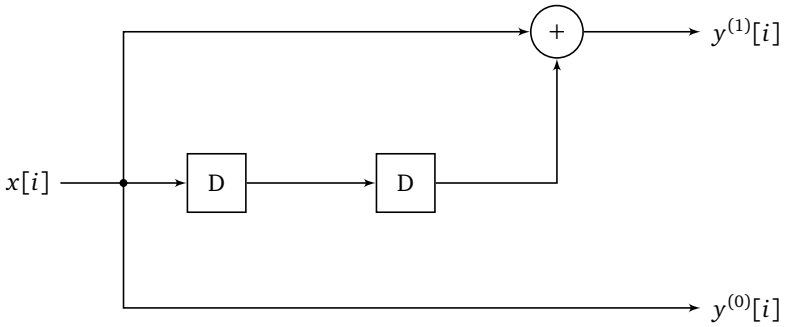


Figure 2.4: Example of a non-recursive convolutional code with asymptotic rate $R_\infty = 1/2$. $x[i]$ is the input and $y[i]$ is the output.

The generator polynomial of this example is given by

$$G^{(0)}(D) = 1 \tag{2.7}$$

$$G^{(1)}(D) = 1 + D^2. \tag{2.8}$$

The impulse response of the parity-bit generator is given by

$$g^{(1)}[i] = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

The output is given by the convolution form:

$$y^{(1)}[i] = x[i] * g^{(1)}[i] \tag{2.10}$$

$$= \sum_{l=0}^{2} x[l] g^{(1)}[i-l]. \tag{2.11}$$

The *constraint length* of a convolutional code is $l_c = m_s + 1$ where m_s is the largest degree in $g[i]$. In the example in Figure 2.4, $l_c = 3$.

2.2.4 Low-Density Parity-Check Convolutional Codes

LDPC-CCs or Spatially-Coupled LDPC (SC-LDPC) codes are formed by imposing the above mentioned convolutional structure on LDPC-BCs. They were invented by Alberto Felström and Kamil Zigangirov [13]. These codes are characterized by a sparse infinite-length PCM which has a diagonal structure. The PCM of these codes is constructed by coupling PCMs of LDPC-BCs as given by

$$\mathbf{H}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & & & & \\ & \mathbf{H}_{m_s}(t-1) & \dots & \mathbf{H}_1(t-1) & \mathbf{H}_0(t-1) & & & \\ & & \mathbf{H}_{m_s}(t) & \dots & \mathbf{H}_1(t) & \mathbf{H}_0(t) & & \\ & & & \mathbf{H}_{m_s}(t+1) & \dots & \mathbf{H}_1(t+1) & \mathbf{H}_0(t+1) & \\ & & & & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \begin{matrix} \\ \mathbf{s}(t-1) \\ \mathbf{s}(t) \\ \mathbf{s}(t+1) \\ \\ \end{matrix} \tag{2.12}$$

where the $\mathbf{H}_\mu(t) \in \mathbb{F}_2^{(n-k)\times n}, \mu = 0, \dots, m_s$ are PCMs of different LDPC-BCs of rate $R_\infty = k/n$ for different time instances and m_s is the memory of the code. Hence, the asymptotic rate of the resulting LDPC-CC is $R_\infty = k/n$. $\mathbf{s}(t) \in \mathbb{F}_2^{(n-k)\times 1}$ denotes the syndromes resulting from the parity-check equations. The codewords of such a code have the form

$$\mathbf{x}^T = \begin{bmatrix} \dots & \mathbf{x}(t-1)^T & \mathbf{x}(t)^T & \mathbf{x}(t+1)^T & \dots \end{bmatrix} \tag{2.13}$$

where each $\mathbf{x}(t) \in \mathbb{F}_2^{n\times 1}$. Given the PCM $\mathbf{H}$ and a valid codeword $\mathbf{x}$, the following expression holds:

$$\mathbf{s}(t) = \sum_{\tau=0}^{m_s} \mathbf{H}_\tau(t)\mathbf{x}(t-\tau) \mod 2. \tag{2.14}$$

The equation (2.14) is a convolution representing the convolutional structure of $\mathbf{H}$ in (2.12). Similar to equation (2.11) for convolutional codes.

The bits in the codeword $\mathbf{x}$ are coupled together over a distance called the *constraint length* which is given by $l_c = (m_s + 1)n$ bits.

2.2.5 Termination of Convolutional Codes

In general, LDPC-CCs have codewords and PCMs of infinite length. For packet-based communication networks, however, the whole packet has to be retransmitted in case of incorrect information bits in higher layers. Also, in a wireless medium the channel parameters change over time which requires the encoder to change its code rate on the fly. For the aforementioned reasons, terminated codes are a better choice.

Termination is the process of limiting the coupling length, so that the codewords have finite length. This allows the decoder to stop decoding the current received word if a bit cannot be corrected, thus reducing the decoding complexity. The termination process requires appending *termination bits* to the end of the codeword to ensure that the last m_s parity-check equations of the terminated PCM are fulfilled. Termination also ensures that the encoder returns to an all-zero state before encoding the next codeword. For recursive convolutional codes, the termination bits are determined by solving a system of linear equations in $\mathbb{F}_2$. Whereas for non-recursive convolutional codes, appending a series of bits with value zero is sufficient.

Termination introduces a rate loss because the termination bits do not contain any information. Hence, for the rate calculation of a terminated LDPC-CC, the termination bits are not taken into account. However, the rate loss is compensated by an increase in decoding performance as the termination reduces the CN degrees at the end of the codeword and smaller CN degrees are better.

The PCM of a terminated LDPC-CC is a sub-matrix of the infinitely long PCM of the code (2.12). The terminated PCM has a structure as given by

$$\mathbf{H}_L = \overbrace{\begin{bmatrix} \mathbf{H}_0(0) & & & \\ \mathbf{H}_1(1) & \mathbf{H}_0(1) & & \\ \vdots & \mathbf{H}_1(2) & \ddots & \\ \mathbf{H}_{m_s}(m_s) & \vdots & \ddots & \mathbf{H}_0(L-1) \\ & \mathbf{H}_{m_s}(m_s+1) & \ddots & \mathbf{H}_1(L) \\ & & & \vdots \\ & & & \mathbf{H}_{m_s}(L+m_s) \end{bmatrix}}^{Ln} \Bigg\} (L+m_s)(n-k) \tag{2.15}$$

where L is the *coupling length* denoting the number of Code Blocks (CBs) in the codeword. Each CB contains n bits. Hence, the total length of the terminated codeword is $n_L = Ln$ bits. The effect of termination in the Tanner graph of a $R_\infty = 1/2$ code is shown in Figure 2.5.

The entire graph in Figure 2.5 can be seen as a Tanner graph of an infinitely long LDPC-CC. As a result of termination, only the center part of the graph remains. The dark circles are the VNs of the terminated LDPC-CC and the solid lines are their corresponding edges.

2.2.6 Low-Density Parity-Check Convolutional Code Used in IEEE 1901

In this thesis, we use the LDPC-CCs specified in the BPL or IEEE 1901 [6] standard to evaluate our decoder. From now on, we refer to the LDPC-CCs in the IEEE 1901 standard as *BPL codes*. The BPL codes are specified as sets of parity-check polynomials for all asymptotic rates $R_\infty = k/n,\ n \in \{2,3,4,5\}$ where $k = n-1$. In other words, the BPL codes have only one parity bit in each CB.

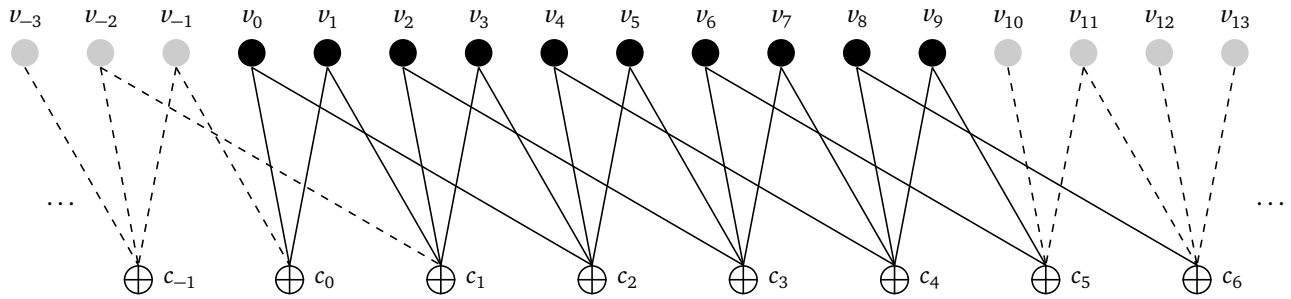


Figure 2.5: Tanner graph of a terminated LDPC-CC. The dark circles and lines are the VNs and edges of the terminated code. The light circles and dashed lines are the omitted VNs and edges as a result of termination.

The codes are defined as

$$\sum_{i=1}^{k} A_{i,\tau}(D)M_i(D) + \sum_{i=1}^{n-k} C_{i,\tau}(D)B_i(D) \equiv 0 \mod 2 \tag{2.16}$$

where k is the number of message bits in each CB, $\tau \in \{0,\ldots,T-1\}$ is the phase of the code that is given by $\tau = (t \bmod T)$, T is the periodicity of the codes, $M_i(D), i = 1,\ldots,k$ represents message bits and $B_i(D), i = 1,\ldots,n-k$ represents parity bits. $A_{i,\tau}$ and $C_{i,\tau}$ are the generator-polynomials that define the connections between the bits based on delay D.

The memory m_s of the code is

$$m_s = \max\left(\{\deg(A_{i,\tau}(D)) : i = 1,\ldots,k; \forall\tau\} \cup \{\deg(C_{i,\tau}(D)) : i = 1,\ldots,n-k; \forall\tau\}\right) \tag{2.17}$$

where $\deg(f(x))$ denotes the set of all degrees of x in $f(x)$.

The BPL codes are periodic with $T = 3$. Periodic codes have time-varying parity-check polynomials which repeat every T CBs. For illustration, the parity-check polynomial of the BPL code for $R_\infty = 2/3$ and $\tau = 0$ is given as

$$(D^{214} + D^{185} + 1)M_1(D) + (D^{194} + D^{67} + 1)M_2(D) + (D^{215} + D^{145} + 1)B(D) = 0 \mod 2, \tag{2.18}$$

for $\tau = 1$ as

$$(D^{160} + D^{62} + 1)M_1(D) + (D^{226} + D^{209} + 1)M_2(D) + (D^{206} + D^{127} + 1)B(D) = 0 \mod 2, \tag{2.19}$$

and for $\tau = 2$ as

$$(D^{196} + D^{143} + 1)M_1(D) + (D^{115} + D^{104} + 1)M_2(D) + (D^{211} + D^{119} + 1)B(D) = 0 \mod 2. \tag{2.20}$$

$m_s = 215$ for $n = 2$ and $m_s = 226$ for $n \neq 2$. All $A_{i,\tau}$ and $C_{i,\tau}$ have three taps for each bit in the CB. There is a maximum of $3n$ taps or edges per CN.

[6] specifies that termination is achieved by appending bits with value 0 to the end of the message bits before encoding. These bits are called *zero-tail bits*. The number of zero-tail bits n_z depends on the number of message bits n_m in the codeword and the asymptotic rate R_∞. The number of CBs in the terminated codeword is

$$L = \frac{n_m + n_z}{n-1}. \tag{2.21}$$

Since the zero-tail bits are known at the receiver, they are not transmitted. Only the parity bits generated from the zero-tail bits are transmitted. Hence, the actual rate of the terminated BPL code is

$$R_L = \frac{n_m}{Ln - n_z}. \tag{2.22}$$

Let us rewrite (2.22) as

$$R_L = \frac{n_m}{L_m n + n_t} \tag{2.23}$$

where L_m is the number of CBs in unterminated codeword and n_t is the number of parity bits in the termination sequence. Also,

$$L = L_m + n_t. \tag{2.24}$$

By rearranging (2.23), the relation between R_L and R_∞ is given by

$$R_L = \nu R_\infty \tag{2.25}$$

where $\nu = \frac{L_m n}{L_m n + n_t}$ and $R_\infty = \frac{n_m}{L_m n} = \frac{k}{n}$. Hence, from (2.24)

$$\lim_{L\to\infty} R_L = R_\infty. \tag{2.26}$$

Encoding and termination of BPL codes is explained in detail in Chapter 3.

2.3 Decoding of Low-Density Parity-Check Codes

A channel decoder attempts to find the transmitted codeword $\mathbf{x}$ from the received word $\mathbf{y}$. The best decoder in terms of performance is a Maximum-A-Posteriori (MAP) decoder. Its complexity grows exponentially with the information word length because it finds—among all possible codewords—the codeword that has the highest probability given the received word. The estimate of the transmitted codeword from a MAP decoder is given by

$$\begin{aligned}\widehat{\mathbf{x}} &= \underset{\mathbf{x}_i}{\operatorname{argmax}}\, p_{X|Y}(\mathbf{x}_i, \mathbf{y}) \\ &= \underset{\mathbf{x}_i}{\operatorname{argmax}}\, \frac{p_{Y|X}(\mathbf{y}, \mathbf{x}_i)\Pr(\mathbf{x}_i)}{\Pr(\mathbf{y})} \\ &= \underset{\mathbf{x}_i}{\operatorname{argmax}}\, p_{Y|X}(\mathbf{y}, \mathbf{x}_i)\Pr(\mathbf{x}_i)\end{aligned} \tag{2.27}$$

where $\mathbf{x}_i$ is a codeword from the set of all codewords, $\mathbf{y}$ is the received word, Y and X are random variables representing received word and transmitted codeword respectively. For equiprobable codewords $\mathbf{x}_i$, a MAP decoder is equivalent to a Maximum-Likelihood (ML) decoder.

Due to the high complexity of MAP decoders, LDPC codes are usually decoded using iterative Message-Passing Algorithm.

2.3.1 Belief Propagation

The Message-Passing Algorithm (MPA) uses the BP technique [14] to compute the *A-Posteriori Probability (APP)* of the bits in the transmitted codeword given the received word, in an iterative fashion. The idea behind belief propagation is exchanging uncertainties between the bits which are connected as defined by the encoder or PCM. Refer to Section 3.1 to see how different bits in the codeword are dependent on each other. The algorithm uses Log-Likelihood Ratios (LLRs) instead of APPs as in (2.27) for numerical stability. The LLR values given by the channel for the received bits are

$$\mathcal{L}(y_i) = \log \frac{\Pr(X_i = 1 \mid \mathbf{y})}{\Pr(X_i = 0 \mid \mathbf{y})} \tag{2.28}$$

where $i = 0, \ldots, n-1$ is the index of bits in the codeword.

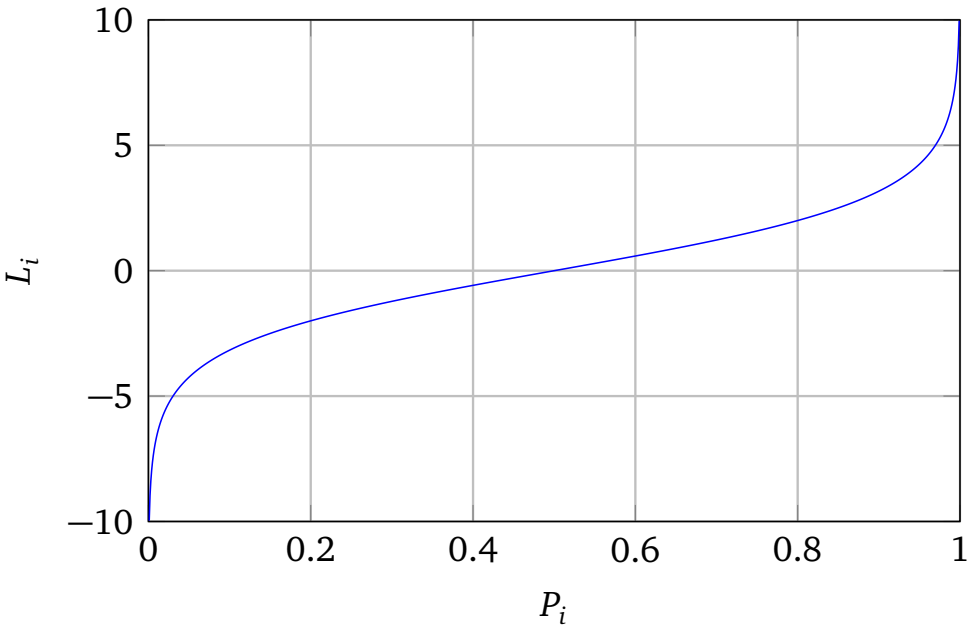


Figure 2.6: Relation between APP and LLR.

Figure 2.6 shows the relation between $\Pr(X_i = 0 \mid \mathbf{y})$ and $\mathcal{L}(y_i)$. The APP values range from 0 to 1 while the LLR takes values ranging from $-\infty$ to $+\infty$ which makes calculation of messages easier.

In a single iteration of the algorithm, the LLRs of each bits in the codeword are updated through two intermediate message computations: variable-node–to–check-node (V2C) message and check-node–to–variable-node (C2V) message.

- V2C message: Each VN passes its LLRs on to its neighboring nodes (neighboring nodes are the CNs to which the VN is connected in the Tanner graph). These LLRs contain only extrinsic information from all other CNs in the previous iteration. The expression for the V2C message is given by [14]

$$\mathcal{L}_{ij}^{\text{vc}} = \mathcal{L}(y_i) + \sum_{j' \in \mathcal{E}_v(i) \setminus j} \mathcal{L}_{j'i}^{\text{cv}} \tag{2.29}$$

  where $\mathcal{L}_{ij}^{\text{vc}}$ is the V2C message from the i-th VN to the j-th CN, $\mathcal{L}_{ji}^{\text{cv}}$ is the C2V message from the j-th CN to the i-th VN in the previous iteration and $\mathcal{E}_v(i)$ is the set containing all n_c CNs connected to the i-th VN.

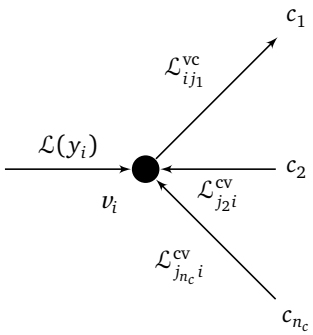


Figure 2.7: Example showing that messages from all other CNs sum-up with the channel's LLR to form the V2C message to the first CN.

- C2V message: Each CN processes the received V2C messages and computes extrinsic information for its neighboring VNs. These extrinsic informations contain V2C messages from VNs other than the destination VN. The expression for C2V messages is given by [14]

$$\mathcal{L}_{ji}^{\text{cv}} = 2\,\text{atanh}\left(\prod_{i' \in \mathcal{E}_c(j)\setminus i} \tanh\left(\frac{\mathcal{L}_{i'j}^{\text{vc}}}{2}\right)\right) \tag{2.30}$$

  where $\mathcal{E}_c(j)$ is the set containing all n_v VNs connected to the j-th CN.

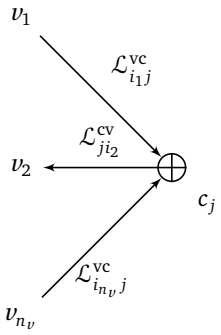


Figure 2.8: Example showing that messages from all other VNs combine using equation (2.30) to form the C2V message to the second VN.

The process of sending a V2C message, receiving a C2V message from the same edge, and summing it up with the current LLR is termed an *edge update*. The above steps indicate the BP technique. It is also called Sum-Product algorithm (SPA).

The high complexity C2V message computation can be approximated by a low-complexity computation called the *Min-Sum Algorithm*. The Min-Sum Algorithm (MSA) version of the expression in (2.30) is given by

$$\mathcal{L}_{ji}^{\text{cv}} \approx \left(\prod_{i' \in \mathcal{E}_c(j)\setminus i} \text{sign}\left(\mathcal{L}_{i'j}^{\text{vc}}\right)\right) \cdot \min_{i'} |\mathcal{L}_{i'j}^{\text{vc}}|. \tag{2.31}$$

One can perform the edge updates in different sequences. Two such sequencing methods are *parallel scheduling* and *serial scheduling*. In parallel scheduling, which is also referred to as *flooding*, the VNs send V2C messages to all CNs at once and then C2V messages are computed and passed to all VNs. In

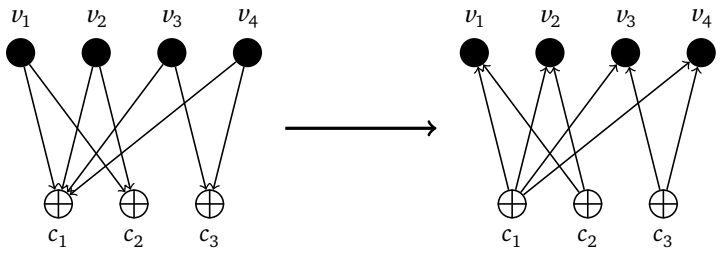


Figure 2.9: Illustration of parallel scheduling with an example Tanner graph.

Figure 2.9, the left graph indicates V2C message passing and the right graph indicated C2V message passing of a flooding schedule. In serial scheduling [15], the VNs are updated in a CN-by-CN or row-by-row (in the PCM) manner. Each CN processes its incoming V2C messages and sends the corresponding C2V messages to its neighboring VNs. This is called a *row update* or a *layer update*. Figure 2.10 shows how message passing is performed in serial scheduling. Parallel scheduling is much faster on a parallel computing platform because the edge updates are independent to each other. But compared to its serial counterpart, parallel scheduling lacks performance because the edges are updated in parallel and only the original messages from the VNs are used. Serial scheduling yields better performance than parallel scheduling as the CNs that are being processed later will have updated LLRs from the VNs. In a conventional BP decoder with any type of scheduling, the entire message passing is repeated for a maximum of I iterations.

With serial scheduling, different orders of CN processing results in decoding performance changes. Figure 2.10 illustrates serial scheduling with CN processing from left to right i.e., top to bottom in the PCM. For irregular codes, a good choice is to start processing the CN that has the lowest r and move to higher r CNs [16].

In our implementation, we use an improved MSA as proposed in [17] which is a combination of (2.30) and (2.31). We do serial scheduling with layer updates starting from top to bottom. We also do a bottom to top layer update which is discussed in Chapter 4.

2.3.2 LDPC-CC-specific Decoding Techniques

The conventional BP-based block decoder can be used to decode any LDPC-BC or terminated LDPC-CC in which the BP is performed throughout the whole Tanner graph at once. For LDPC-CCs, the convolutional structure imposes a constraint on the VNs: two VNs of the PCM that are at least $(m_s+1)n$ columns apart cannot be involved in the same parity-check equation. This characteristic can be exploited to perform iterative BP decoding only to a part of the codeword at a time. Two of such decoding techniques are *pipeline* and *windowed* decoding.

Pipeline Decoder

A decoding technique that was proved to be efficient for LDPC-CCs is the *pipeline decoding* introduced in [13]. The pipeline decoder employs I parallel processing units. Each processing unit covers $l_c = (m_s + 1)n$ VNs, so that during a single decoding iteration the messages are only passed to other VNs within the same processing unit. Hence, all the processing units cover a total of Il_c VNs. In each time instance, n VNs and $n-k$ VNs enter the rightmost processor and n estimated VNs leave from the leftmost processor. Hence, at the end of Il_c time instances, all CNs are processed I times. Figure 2.11 illustrates pipeline decoding of a code with $R_\infty = 1/2$, $m_s = 3$ and $I = 3$. The codeword is indicated by the rectangular bar above the PCM. The red windows are the three processing units. The green (backhatched) part of the codeword indicates the estimated VNs and the blue (hatched) part indicates

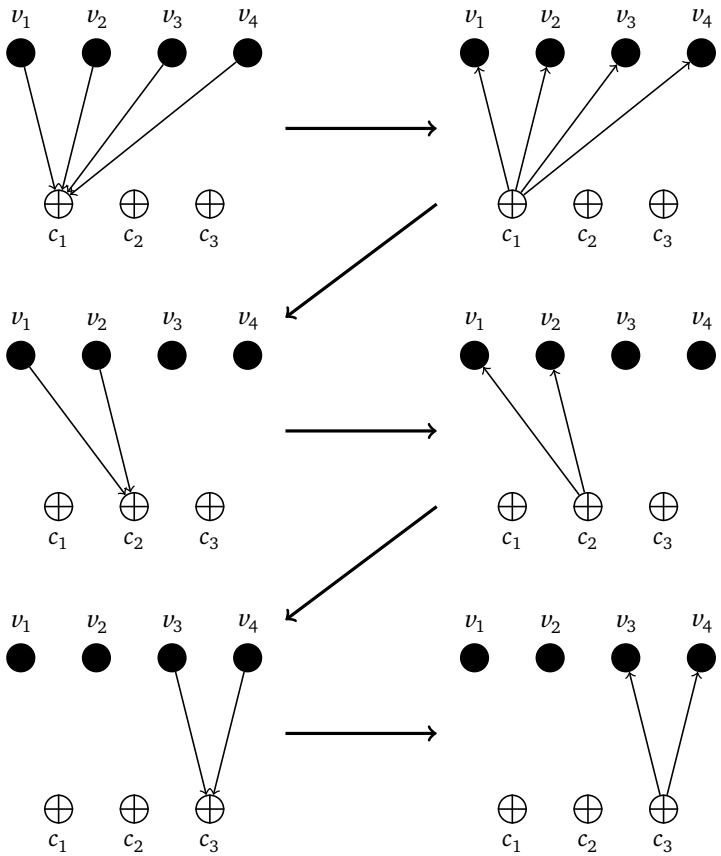


Figure 2.10: Illustration of serial scheduling with the same Tanner graph as in Figure 2.9. The processing starts from the left most CN and moves right to the last CN.

the next available estimated VNs. The brown (vertically hatched) and red (horizontally hatched) regions indicate the VNs that are currently being processed and VNs that are yet to be processed, respectively. The arrows indicate the CNs that are processed at the current time instance.

The decoding speed (the output bit rate of the decoder) of a pipeline decoder is given by

$$\kappa_{\mathrm{p}} = \frac{n}{\psi} \tag{2.32}$$

where ψ is the time taken to perform one CN processing.

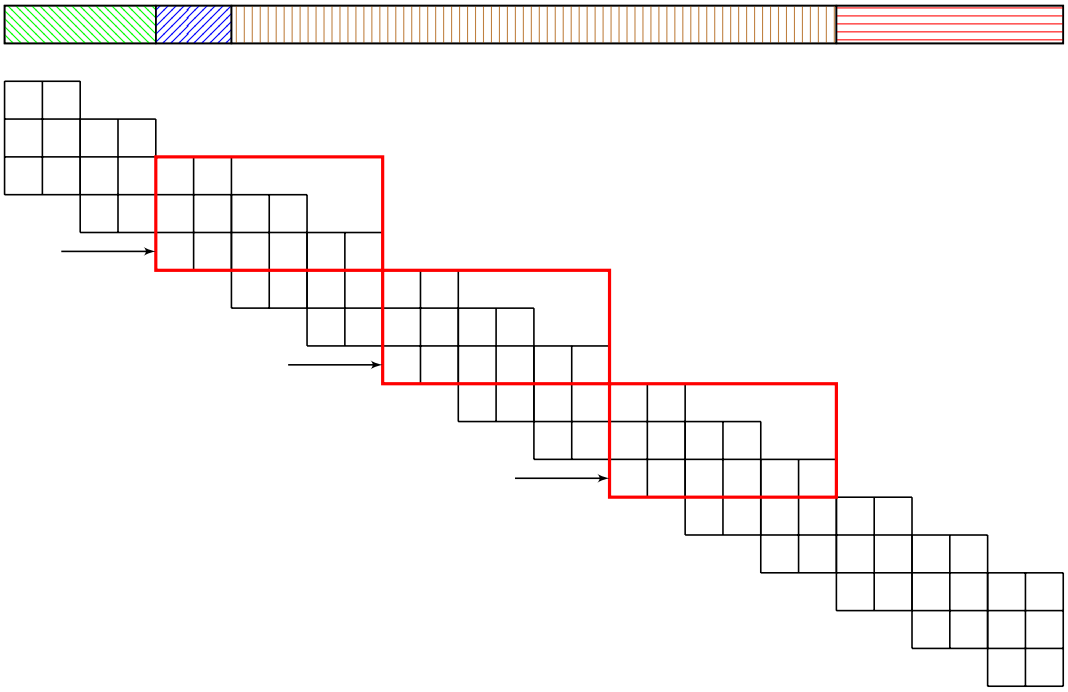


Figure 2.11: PCM illustrating the pipeline decoding technique for code with $R_\infty = 1/2$, $m_s = 2$ and $I = 3$. The arrows indicate the CNs that are processed at the current time instance.

Windowed Decoder

Figure 2.12 illustrates a windowed decoder with window size of $W = 4$ and window position $\rho = 5$. The solid red box is the current window instance and the red dashed box indicates the next window instance. The blue hatched region of the codeword is the *target VNs*, the red vertically hatched region is rest of the VNs that are updated inside the current window. The blue colored edges in the PCM are the edges that are updated during the current window instance. The red colored edges are outside the window but they are still updated since their corresponding CNs are inside the window. The green backhatched region of the codeword are the VNs that receive updates from the red colored edges.

One may choose to not update the edges that are outside the window. This is to prevent the correctly estimated LLR values of the VNs from updating to incorrect values that might propagate from inside the window. Another option might be to only read the LLRs values from VNs connected to the red edges. This prevents the estimated LLR of the VNs from updating to incorrect values while having the advantage of using their LLRs to update the VNs inside the window. In this thesis, we choose to update the VNs in the green backhatched region. This has an advantage of increasing the magnitude of their LLRs which in turn has a positive influence on the VNs in the next window instances. Also, the VNs in the left of the window are more reliable than in the right of the window. Hence, the probability of LLR of the

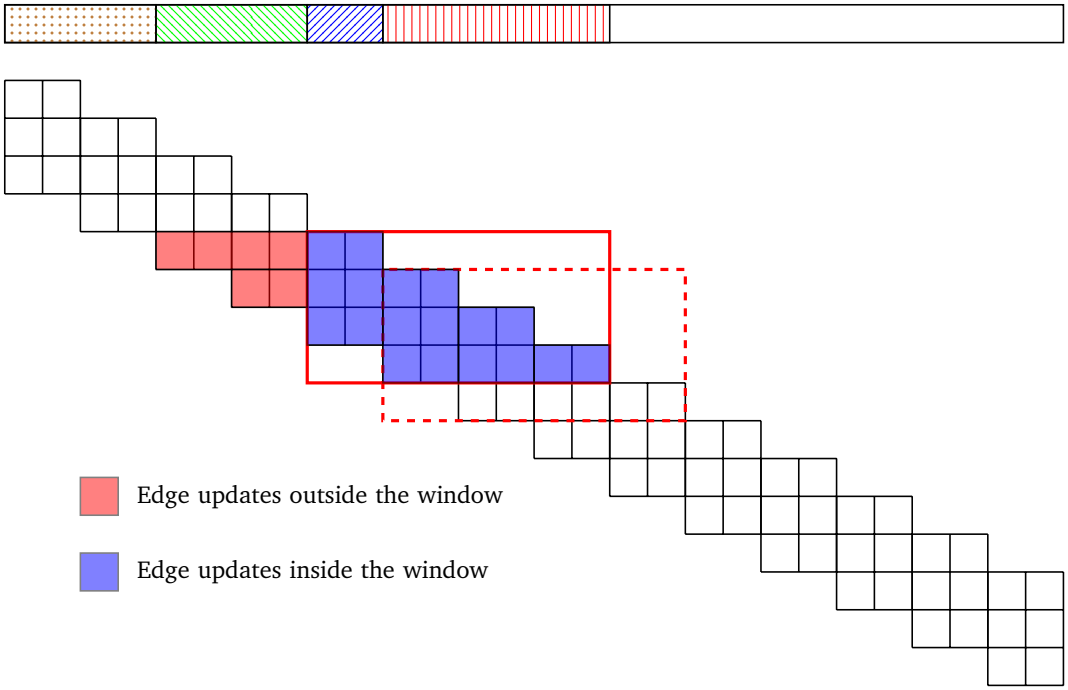


Figure 2.12: PCM illustrating the windowed decoder for code with $R_\infty = 1/2$ and $m_s = 2$. The current window position $\rho = 5$ is indicated by the solid red box and the next window position is indicated by the dashed red box.

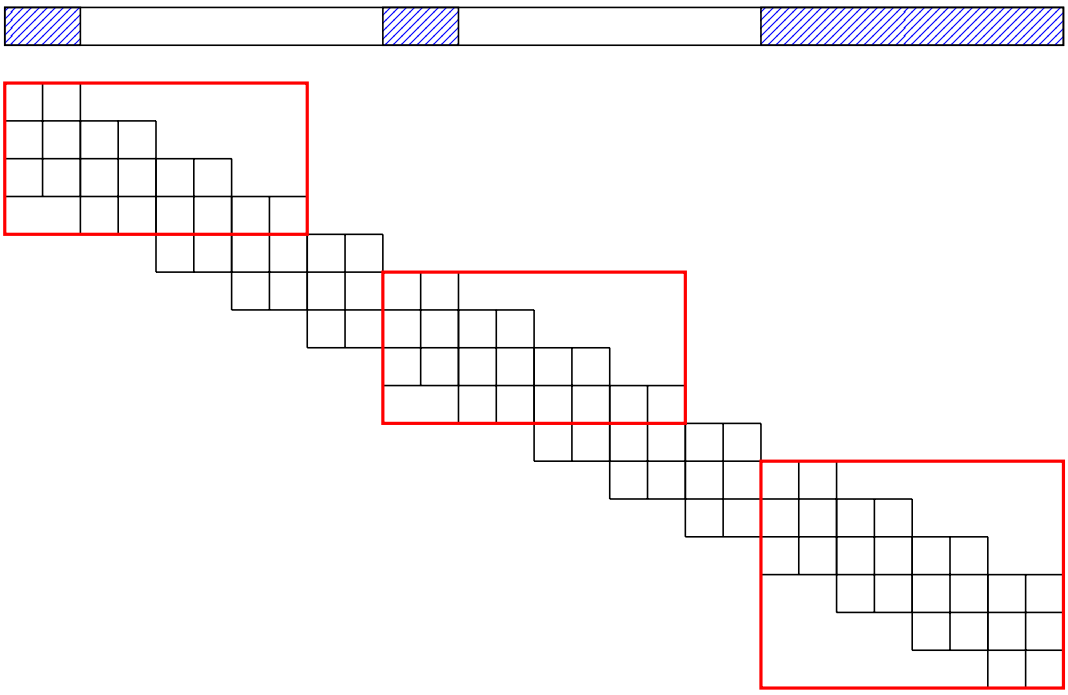


Figure 2.13: Target VN for first, middle and the last window positions.

VNs in the green backhatched region flipping their sign is low. This motivates us to choose this window configuration. The brown dotted region of the codeword is the VNs that are already estimated and no longer receive updates from further decoding.

A window must include Wn VNs and $W(n-k)$ CNs. Hence, the size of the window should be $W \geq (m_s+1)$ because smaller window sizes will not include at least one full CN with all its edges. At each window instance, the first n VNs are considered to be the target nodes. However, all VNs connected to the $W(n-k)$ CNs are updated, only the correctness of target nodes are considered as the criteria for moving on to the next window. The BP decoding is performed within the current window for a maximum number of I iterations or until the parity-check equations involving the target nodes are fulfilled (whichever occurs earlier). Then the window shifts forward such that the next n VNs become the target nodes. The process continues until all VNs in the received word are decoded.

With the windowed decoder, there are several ways to perform the decoding when the window reaches the rightmost end of the PCM. The conventional way is to keep moving the window until the last n VNs are the target VNs. Another method is when the window touches the rightmost column of the PCM, the window extends its height to include the remaining m_s CNs of the PCM. Also, all the VNs inside the last window are considered as target VNs. The last window scenario is illustrated in Figure 2.13. We choose the second method as it reduces the number of iterations compared to the conventional method. The lower CN degree in the end of the PCM compensates to the reduced decoding performance for having fewer iterations.

The decoding speed of a windowed decoder is given by

$$\kappa_{\mathrm{w}} = \frac{n}{\psi \cdot I}. \tag{2.33}$$

The main benefit of windowed decoding over full-block decoding is that the memory requirements are reduced because at any instance, the BP is performed on a smaller number of VNs rather than the whole graph. For non-packet-wise transmissions, the decoded bits are sent to higher layer for processing rather than waiting for the whole codeword to be decoded as in a block decoder. However, these benefits come with a cost of reduced performance since the BP is limited to fewer VNs and CNs.

In this thesis, we choose window decoding over pipeline decoding because implementing parallel processing units in software is infeasible.

2.4 System Model

In this thesis, we consider a digital baseband system model for simulations as shown in Figure 2.14. The information bits are generated using a random-number generator with uniform distribution. The sequence of bits $\mathbf{m}$ of the generated random bit is encoded using the *BPL Codes Encoder* block. The details of this block are discussed in Chapter 3. The *Quadrature Amplitude Modulation (QAM) Modulator* block receives the codewords $\mathbf{x} \in \mathbb{F}_2^{Ln\times 1}$ from the encoder and maps them to complex-valued symbols depending on the chosen modulation scheme. The output of the modulator is a vector of symbols given by $\mathbf{u} \in \mathbb{C}^{\frac{Ln}{o}\times 1}$ where o is the order of modulation and lets assume that $Ln = go, g \in \mathbb{Z}_+$. Figure 2.15 shows a Quadrature Phase Shift Keying (QPSK) symbol constellation. It is seen that the adjacent symbols differ by only one bit. This is called *Gray Mapping* [10]. In all our simulations we use QPSK with gray mapping and no interleaving. Hence, $\mathbf{u}_i, i = 0, \ldots, \frac{Ln}{o} - 1 \in \{e^{j\frac{\pi}{4}}, e^{j\frac{3\pi}{4}}, e^{-j\frac{3\pi}{4}}, e^{-j\frac{\pi}{4}}\}$.

The channel model we consider is a simple Additive White Gaussian Noise (AWGN) channel with no fading or multi-path components. So the received symbols are given by

$$\mathbf{v} = \mathbf{u} + \mathbf{z} \tag{2.34}$$

where $\mathbf{z} = \mathcal{CN}(0, 2\sigma^2) \in \mathbb{C}^{\frac{Ln}{o}\times 1}$ is a complex random variable of Gaussian distribution with zero mean

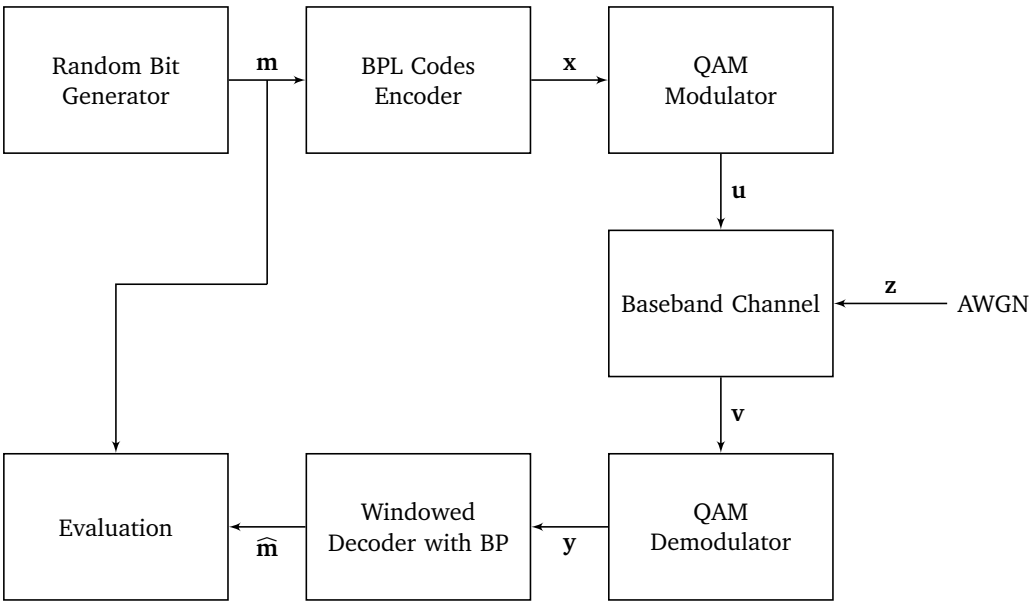


Figure 2.14: System model for simulations.

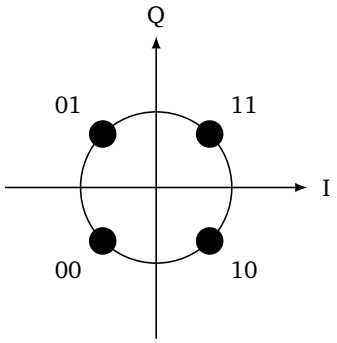


Figure 2.15: Symbol constellation of QPSK modulation with gray mapping.

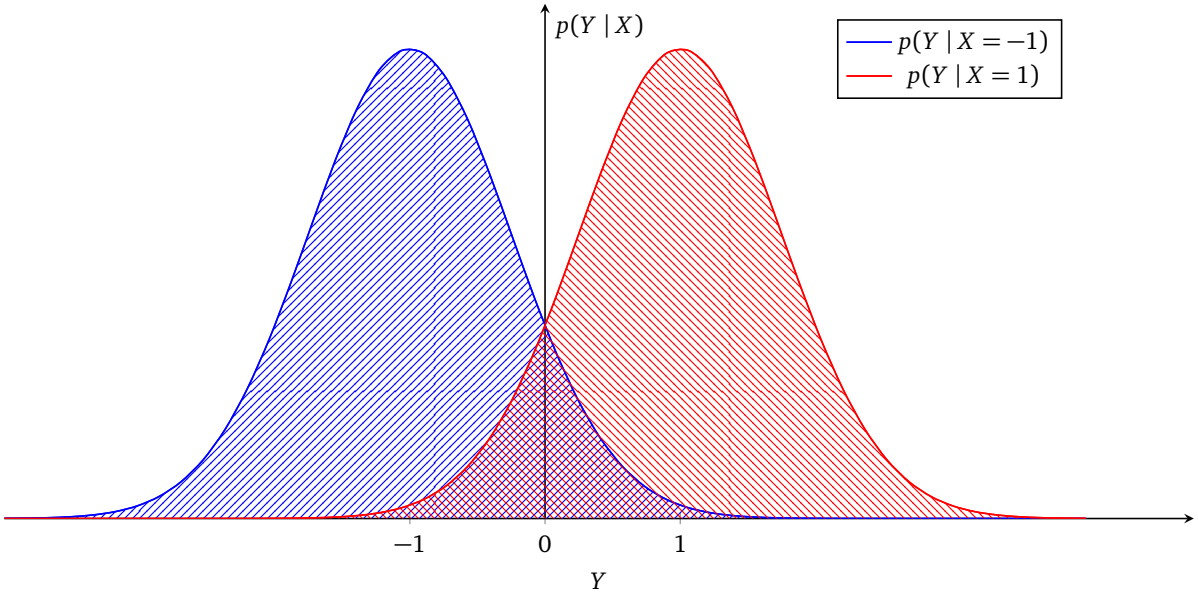


Figure 2.16: Conditional probability density functions of a BPSK symbol over AWGN channel.

and variance $2\sigma^2$. Note that only one symbol in $\mathbf{u}$ is transmitted per channel use. The received symbol vector is $\mathbf{v} \in \mathbb{C}^{\frac{Ln}{o} \times 1}$. The output of the *QAM Demodulator* is a vector of LLRs $\mathbf{y} \in \mathbb{R}^{Ln \times 1}$ corresponding to the bits in $\mathbf{x}$. The LLRs are computed using the symbols in $\mathbf{v}$ and the modulation order. Each QPSK symbol is a combination of two BPSK symbols that are orthogonal to each other. Hence, the LLRs of both the bits can be computed separately. The LLR of the first bit of each received QPSK symbol y is computed as

$$\mathcal{L}_{\text{first}} = \log \frac{\Pr(X_1 = 0 \mid \text{Re}(y))}{\Pr(X_1 = 1 \mid \text{Re}(y))} \tag{2.35}$$

and of the second bit as

$$\mathcal{L}_{\text{second}} = \log \frac{\Pr(X_2 = 0 \mid \text{Im}(y))}{\Pr(X_2 = 1 \mid \text{Im}(y))}. \tag{2.36}$$

Using Bayes' Theorem and assuming that the bits in $\mathbf{x}$ are equiprobable, we have

$$\mathcal{L}_{\text{first}} = \log \frac{p_{\text{Re}(Y)|X}(\text{Re}(y), X_1 = 0)}{p_{\text{Re}(Y)|X}(\text{Re}(y), X_1 = 1)}, \tag{2.37}$$

$$\mathcal{L}_{\text{second}} = \log \frac{p_{\text{Im}(Y)|X}(\text{Im}(y), X_2 = 0)}{p_{\text{Im}(Y)|X}(\text{Im}(y), X_2 = 1)}. \tag{2.38}$$

On substituting the conditional Probability Density Functions (pdfs) of BPSK symbols as shown in Figure 2.16, we get

$$\mathcal{L}_{\text{first}} = \log \frac{e^{-\frac{(\text{Re}(y)-1)^2}{2\sigma^2}}}{e^{-\frac{(\text{Re}(y)+1)^2}{2\sigma^2}}} \tag{2.39}$$

$$= \frac{2}{\sigma^2} \text{Re}(y). \tag{2.40}$$

Similarly,

$$\mathcal{L}_{\text{second}} = \frac{2}{\sigma^2} \text{Im}(y). \tag{2.41}$$

Note that the symbol values in Figure 2.16 are $X_i \in \{-1, +1\}, i = 1, 2$ which corresponds to information bits 1 and 0 respectively. The noise variances of the real and imaginary parts of the complex random variable Z are each σ^2 from

$$\text{E}\{||Z||^2\} = \text{E}\{||\text{Re}(Z)||^2\} + \text{E}\{||\text{Im}(Z)||^2\}. \tag{2.42}$$

The Signal-to-Noise Ratio (SNR) ζ of the QPSK signal is

$$\frac{E_s}{N_0} = 2\frac{E_b}{N_0} \tag{2.43}$$

where E_s is the energy per QPSK symbol, E_b is the energy per bit and N_0 is the AWGN power spectral density. The SNR of each bit is equivalent to the SNR of BPSK signal and is given by

$$\frac{E_b}{N_0} = \frac{\mathrm{E}\{||X_1||^2\}}{\mathrm{E}\{||\,\mathrm{Re}(Z)||^2\}} \tag{2.44}$$

$$= \frac{\mathrm{E}\{||X_2||^2\}}{\mathrm{E}\{||\,\mathrm{Im}(Z)||^2\}} \tag{2.45}$$

$$= \frac{1}{\sigma^2}. \tag{2.46}$$

Hence, the LLRs can be written as

$$\mathcal{L}_{\text{first}} = 2\frac{E_b}{N_0}\,\mathrm{Re}(y), \tag{2.47}$$

$$\mathcal{L}_{\text{second}} = 2\frac{E_b}{N_0}\,\mathrm{Im}(y). \tag{2.48}$$

After the LLRs are computed, they are decoded using the BP and windowed decoding techniques used inside the *windowed decoder with BP* block. The decoding-improvement techniques that are used in this block are discussed in Chapter 4. The estimated bits $\widehat{\mathbf{m}}$ are then compared with the output of the random bit generator $\mathbf{m}$ to calculate the BER and Block-Error Rate (BLER). BER P_b is the ratio between the number of error-bits and the total number of bits transmitted. The probability of error for each bit in the codeword is given by

$$P_b(i) = \Pr\{\widehat{X}_i \neq X_i\} \tag{2.49}$$

and the overall probability of bit error is give by

$$P_b = \frac{1}{Ln}\sum_{i=0}^{Ln-1} P_b(i) \tag{2.50}$$

BLER P_L is the ratio between the number of error-blocks and the total number of blocks transmitted. A block is considered to be an error-block if at least one information bit is incorrect. The BLER is given by

$$P_L = \Pr\{\widehat{\mathbf{m}} \neq \mathbf{m}\}. \tag{2.51}$$

With BER and BLER being the metrics for measuring decoding performance, Total Number of Edge Updates (TNEU) is the metric for measuring decoding complexity. TNEU η is given by

$$\eta = \sum_{\rho=0}^{\xi-1} \eta_\rho I_\rho \tag{2.52}$$

where ξ is the total number of window positions, η_ρ is the number of edges inside and I_ρ is the number of iterations performed at ρ-th window position.

3 Encoding of the Broadband Power Line Codes

In this chapter, we discuss how the encoder for BPL codes is designed and how termination is handled. In some places of this chapter, we use examples of LDPC-CCs with small coupling length L and syndrome-former memory m_s as the BPL codes are too large to be represented legibly on paper.

3.1 Encoder Design

Encoding algorithms are not as complex as decoding algorithm because decoders need to correct the incorrect bits in the codeword. The encoder for any Convolutional code or LDPC-CC only needs to generate the parity bits from the information bits and previously generated parity bits. From equation (2.16), the parity-check term of BPL codes is given by

$$\sum_{i=0}^{m_s} \mathbf{h}_{M,i}\mathbf{m}(t-i) + \sum_{i=0}^{m_s} h_{B,i}\, b(t-i) \mod 2 \tag{3.1}$$

where $\mathbf{m}(t) \in \mathbb{F}_2^{k\times 1}$ is a vector of message bits at t-th time instance or CB, $h_{B,i} \in \mathbb{F}_2$ is the coefficient of polynomial of the parity bit. $\mathbf{h}_{M,i} \in \mathbb{F}_2^{1\times k}$ is a vector of coefficients of polynomials of message bits and is given by (3.3). The encoder generates one parity bit per $n-1$ message bits and the expression for generating the parity bit is given by rearranging (3.1) as

$$b(t) = \sum_{i=0}^{m_s} \mathbf{h}_{M,i}\mathbf{m}(t-i) + \sum_{i=1}^{m_s} h_{B,i}\, b(t-i) \mod 2, \tag{3.2}$$

where $b(t) \in \mathbb{F}_2$ is the parity bit.

$$\mathbf{h}_{M,i} = \begin{bmatrix} [D^i]A_{1,\tau} & \dots & [D^i]A_{k,\tau} \end{bmatrix}, \tag{3.3}$$

where $[D^i]A_{1,\tau}$ represents the coefficient of D^i in polynomial $A_{1,\tau}(D)$.

The resulting codewords $\mathbf{x} \in \mathbb{F}_2^{Ln\times 1}$ have a systematic structure given by

$$\mathbf{x}^T = \begin{bmatrix} \mathbf{x}(0)^T & \mathbf{x}(1)^T & \dots & \mathbf{x}(L-1)^T \end{bmatrix}. \tag{3.4}$$

Similarly, the PCM can be formed using $\mathbf{h}_{M,i}$ and $h_{B,i}$ for all $i = 0, \ldots, m_s$:

$$\mathbf{H} = \begin{bmatrix} [\mathbf{h}_{M,0}, h_{B,0}] & & & & \\ [\mathbf{h}_{M,1}, h_{B,1}] & [\mathbf{h}_{M,0}, h_{B,0}] & & & \\ \vdots & [\mathbf{h}_{M,1}, h_{B,1}] & \ddots & & \\ [\mathbf{h}_{M,m_s}, h_{B,m_s}] & \vdots & \ddots & [\mathbf{h}_{M,0}, h_{B,0}] & \\ & [\mathbf{h}_{M,m_s}, h_{B,m_s}] & \ddots & [\mathbf{h}_{M,1}, h_{B,1}] & \\ & & \ddots & \vdots & \\ & & & [\mathbf{h}_{M,m_s}, h_{B,m_s}] & \end{bmatrix}. \tag{3.5}$$

Encoding of BPL codes is done by generating parity bits as per (3.2). This is performed through the following steps:

1. The information bits are divided into several CBs and copied into the output buffer of the encoder. An example of this step for a code with $R_\infty = 2/3$ is shown in Figure 3.1.

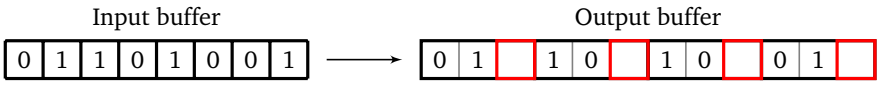


Figure 3.1: Illustration of input and output buffers and its contents. The red boxes represents the position of the parity bits.

2. The parity bits are generated by performing addition modulo 2 with the bits (both information and parity bits) in the output buffer according to (3.2). An example of the parity-bit generation of a code with $R_\infty = 2/3$ is shown in Figure 3.2.

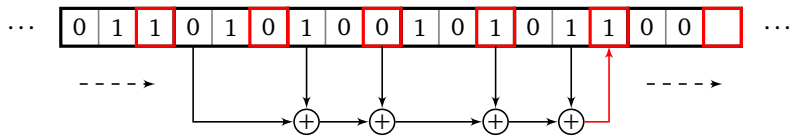


Figure 3.2: Illustration of a parity bit being generated using information and other parity bits in the output buffer. The dashed arrows indicate the direction of movement of the parity-bit generator. Note: An example code.

3. The termination sequence is appended to the output buffer of the encoder. The procedure to generate the termination sequence is discussed in Section 3.2.

3.2 Termination Sequence

The objective of terminating any LDPC-CC is to introduce some low degree CNs in the PCM. Termination essentially means truncating the PCM to have a finite number of rows and columns. Truncating the PCM leads to a condition where the last $m_s n$ bits of the codeword do not satisfy the last m_s parity-check equations of the PCM. To satisfy the last m_s parity-check equations, either the last $m_s n$ or lesser bits need to modified or an appropriate termination sequence $\mathbf{a}$ must be appended to the codeword. Since the last $m_s n$ bits of the codeword should not be modified because they contain information, the later approach is used. The appending of termination sequence is illustrated in Figure 3.3 with a PCM that has a structure similar to (3.5).

3.2.1 Proper Termination

Since the generator polynomials of the BPL codes are recursive, continuously feeding in zero-bits into the encoder after the information bits will not reset the internal states of the encoder. i.e., the output of the encoder will never be all zeros. Hence, one must solve a system of linear equations to find a proper termination sequence that brings the encoder to all-zero state.

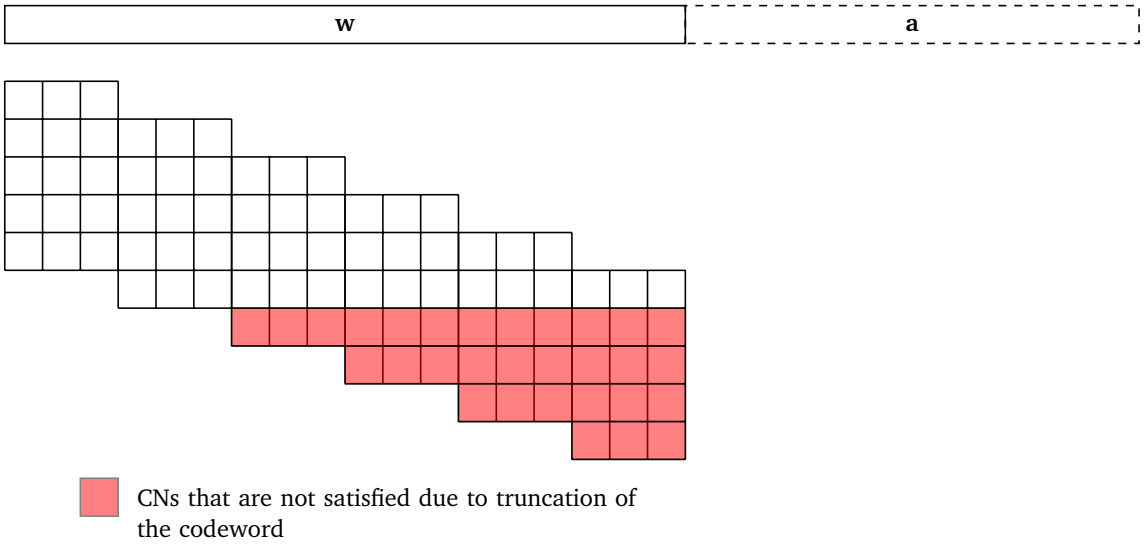


Figure 3.3: PCM illustrating that a truncated codeword do not satisfy all CNs in a $R_\infty = 2/3$ code with $m_s = 4$.

The procedure for determining a termination sequence is reproduced here from [18]. Figure 3.4 illustrates the different matrices and vectors that are used in determining the termination sequence. $\mathbf{e} \in \mathbb{F}_2^{m_s n \times 1}$ is the last part of the codeword vector, vector $\mathbf{a} \in \mathbb{F}_2^{l_t \times 1}$ is the termination sequence that should be determined and appended to the end of $\mathbf{e}$. The sub-matrix $\mathbf{P}$ of the PCM contains the last m_s CNs of the actual PCM. The sub-matrix $\mathbf{D}$ (containing the blue edges) is an extension to the actual PCM due to the termination sequence $\mathbf{a}$. The codeword combined with a proper termination sequence must fulfill all the parity-check rows in matrix $[\mathbf{P}, \mathbf{D}]$. Hence, provided a correct termination sequence $\mathbf{a}$, the following equations from [18] hold true.

$$\begin{bmatrix} \mathbf{P} & \mathbf{D} \end{bmatrix} \odot \begin{bmatrix} \mathbf{e} \\ \mathbf{a} \end{bmatrix} = \mathbf{0} \tag{3.6}$$

$$\mathbf{P} \odot \mathbf{e} + \mathbf{D} \odot \mathbf{a} = 0 \tag{3.7}$$

One possible solution under the constraint that $\mathbf{D}$ is square and invertible is given by

$$\mathbf{a} = \mathbf{D}^{-1} \odot \mathbf{P} \odot \mathbf{e}. \tag{3.8}$$

The maximum length l_t of the termination sequence $\mathbf{a}$ should not exceed the one given in the BPL standard. However, a shorter termination sequence with $l_t \geq n$ can be chosen such that $\mathbf{D}$ is a square matrix. Once the termination sequence $\mathbf{a}$ is determined from (3.8), it is appended to the codeword and the remaining part of the termination sequence (if any) is filled with zeros. These zeros do not contribute to any parity checks. For an encoder that is implemented with feedback shift registers, these zeros are required to bring the encoder to an all-zero state. But, we omit these zeros since we implement the

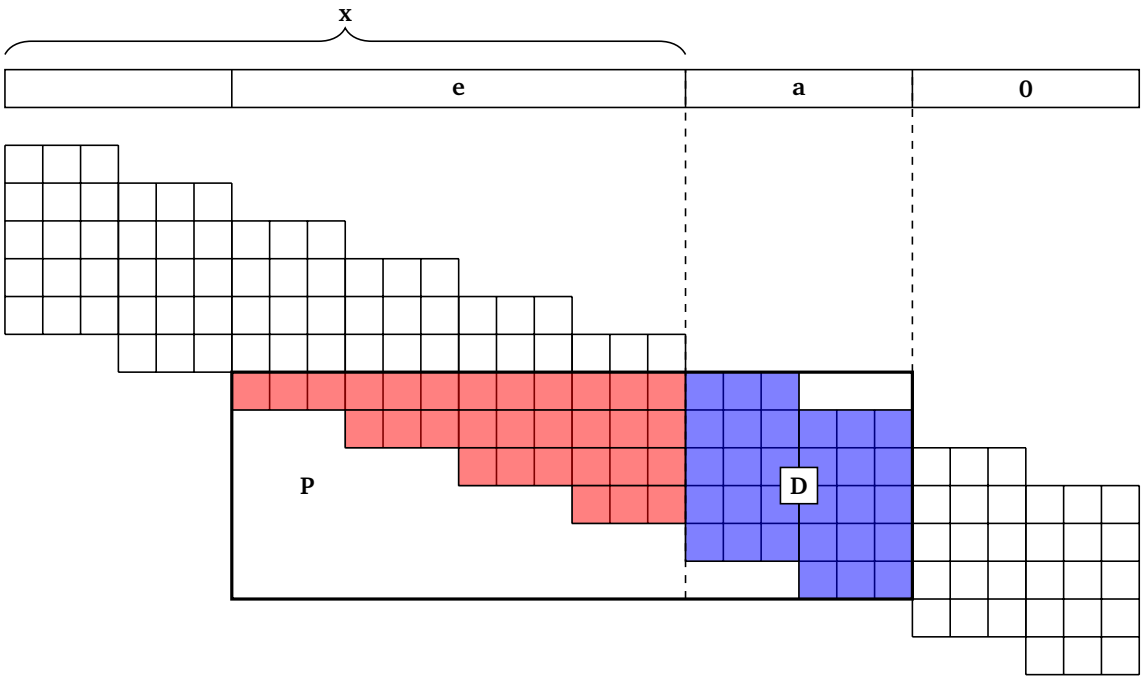


Figure 3.4: PCM of an example LDPC-CC code with $R_\infty = 2/3$ and $m_s = 4$ illustrating sub-matrices used to determine the termination sequence. Depicted above the PCM is the codeword vector.

encoder in software. Hence, the resulting PCM and codeword looks like the one shown in Figure 3.4 with no zeros at the end of the codeword and without their corresponding edges in the PCM.

3.2.2 Zero-Tail Termination

Unfortunately, we found that a proper termination sequence cannot be computed for BPL codes. It is found through numerical evaluations that **D** is not full-rank and hence its inverse does not exist. A full-rank **D** matrix is possible if $\mathbf{D} \in \mathbb{F}_2^{u \times v}$ and $u > v$. But, with the help of numerical solvers, we found that the solution for such an overdetermined system does not exist either. Hence, a termination sequence to satisfy the last m_s parity checks cannot be found for BPL codes. So, we choose to omit the last m_s parity checks from the PCM and do the termination by *zero-tailing* as mentioned in the standard.

The steps for performing zero-tailing termination are as follows.

1. The n_z zero-tail bits are appended to the input buffer of the encoder after the n_m information bits.
2. The encoding is performed as mentioned in Section 3.1 using all the information bits and zero-tail bits.

With the zero-tail termination, the zero-tail bits are always known at the receiver. Hence, the zero-tail bits are not transmitted in an actual system. Only the parity bits generated from the zero-tail bits are transmitted and used during decoding.

The zero-tail termination does not satisfy the last m_s parity checks of the PCM. Hence, the PCM of a zero-tail-terminated BPL codes looks like the one shown in Figure 3.5. The structure of a zero-tail-terminated codeword looks like the one shown in Figure 3.6.

The PCM of the zero-tail-terminated codes has the same CN degrees in the middle and in the end. By contrast, properly terminated codes have lower CN degrees at the end. The lack of low CN degrees reduce the performance of the code as lower CN degree means better reliability of the variable nodes connected to them. However, the zero-tail bits improve the reliability of the connected VNs during the BP decoding process since the zero-tail bits are known at the receiver. The knowledge of the zero-tail

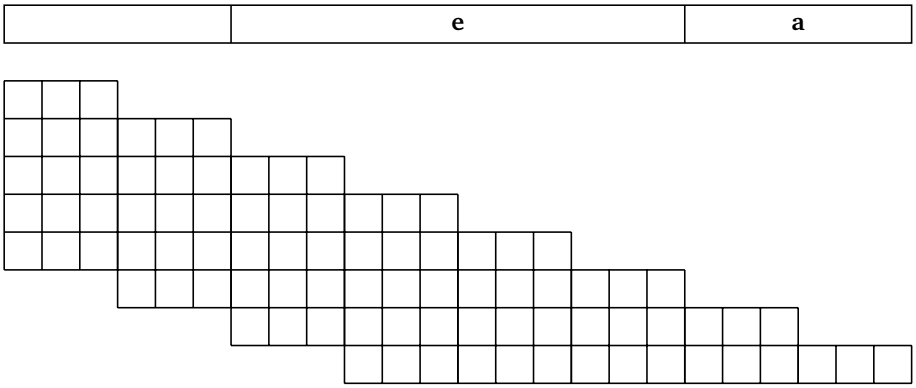


Figure 3.5: PCM and codeword after adding the zero-tail termination sequence.

Figure 3.6: Illustration of a zero-tail-terminated codeword of a code with $R_\infty = 2/3$. Black boxes represent information bits, blue boxes represent zero-tail bits and red boxes represent parity bits.

bits effectively reduce the CN degree in the end-termination of the codeword. Figure 3.7 shows the r over for all CNs in the PCM for code with $R_\infty = 2/3$ and $n_m = 3500$ when zero-tail bits are not known at the receiver. We can see that in the start-termination region, the r varies from 3 to 9 and it remains at 9 through the rest of the codeword. Figure 3.8 shows the r over all CNs when zero-tail bits are known at the receiver. We remove the corresponding columns of the zero-tail bits from the PCM because they are known. This removal of columns effectively reduces the r in the end-termination of the PCM. However, they are not as low as the CN degree in the start-termination.

We conclude this section by summarizing that proper termination for BPL codes are not feasible because of the nature of the PCMs and hence, we perform zero-tail termination. The important differences between these two types of terminations are listed in Table 3.1 and we see that both terminations have advantages and disadvantages over the other.

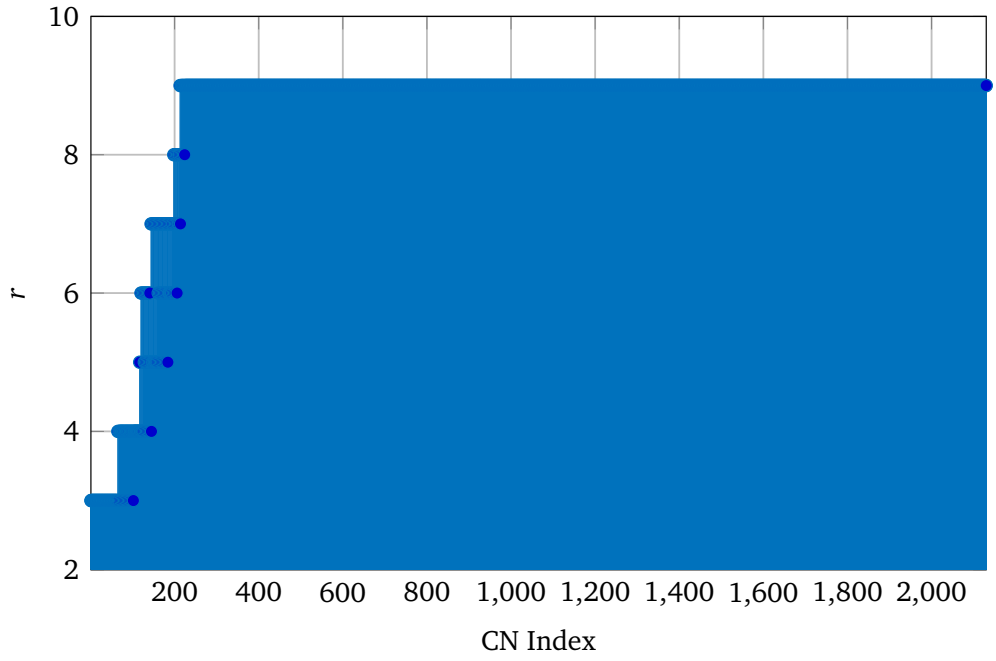


Figure 3.7: r of all CNs in the PCM when zero-tail bits are not known at the receiver.

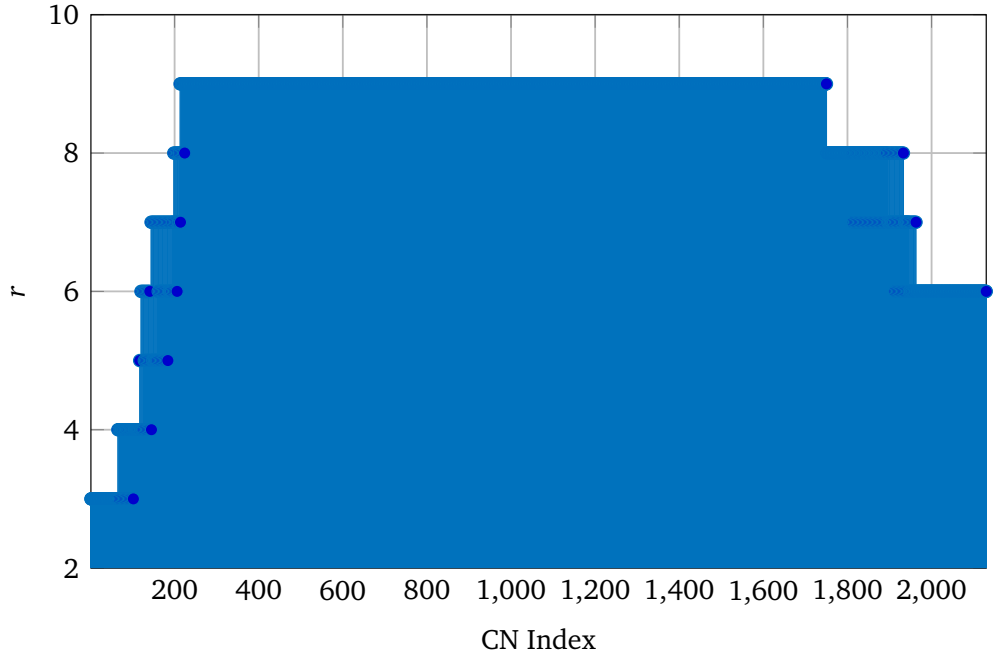


Figure 3.8: r of all CNs in the PCM when zero-tail bits are known at the receiver.

Aspect of Comparison	Proper Termination	Zero-Tail Termination
Omission of CNs	No CNs are omitted from PCM.	Last m_s CNs are omitted from the PCM.
Ease for implementation using shift registers	Encoder returns to all-zero state.	Encoder does not return to all-zero state.
Memory and complexity	Predetermined matrices $\mathbf{F} = \mathbf{D}^{-1} \odot \mathbf{P}$ are stored for each τ and R_∞ to compute $\mathbf{a} = \mathbf{F} \odot \mathbf{e}$. Requires matrix multiplication to be performed.	No storage or matrix multiplication is required. Termination sequence is determined by just using the encoder.
CN degree	Lower CN degree at the end of the PCM than the middle. This enhances decoding performance.	Same CN degree at the end of the PCM as the middle.
Knowledge of termination sequence at receiver.	Termination sequence is unknown at the receiver.	Zero-tail bits are known at the receiver. This effectively reduces the CN degree but not as low as the CN degree in the start-termination.

Table 3.1: Difference between proper termination and zero-tail termination.

4 Decoding Improvements

In this chapter, we discuss the techniques that are developed in this thesis that improve the decoding performance of window decoders. We start with the discussion of already existing techniques that improve the decoding performance and the motivation to develop new techniques. We then discuss the two techniques that are developed in this thesis.

4.1 Literature Overview

There are many aspects used in a windowed decoder and the optimality of each contributes to the increase in decoding efficiency. We choose two of those aspects to analyze and improve them.

4.1.1 Window Direction

The direction of movement of window is an important factor to consider when it comes to performance. In a conventional sliding-windowed decoder, the window moves from left to right of the PCM. This is the optimal direction for a system with continuous encoding and decoding. Since we deal with packet wise transmissions, we opt to consider the idea of moving the window in the reverse direction. Also, in terminated codewords the window direction matters as they have smaller CN degree r at both start and end terminations.

Improvement techniques involving window direction for decoding LDPC-CCs are rarely discussed in literatures. An attractive technique that we found in the literature is the *Zigzag decoder* proposed in [19]. In a conventional window decoder (sliding-window decoder), the window moves from left to right of the PCM [20]. Thus, the information about the probability of the bits flow from left to right of the codeword, i.e., in only one direction. This results in a loss in performance compared to a full-block decoder. The Zigzag decoder moves the window from right to left of the PCM within a small part where the windows did not converge (i.e., the maximum number of iterations have reached and the target CNs did not fulfill the parity checks). It allows the information about probability of the bits to flow from right to left of the codeword, i.e., in the opposite direction. Hence, the different set of information have a positive effect on the reliability of the whole codeword.

4.1.2 Window Convergence

Another improvement technique that we found to be interesting is about the condition for convergence of a window. Although its very natural to stop decoding when the target VN are correct, one should choose an optimum criterion for deciding whether the target VN are correct. A heuristic choice is to check whether the *target CNs*, i.e., the CNs connected to the target VNs fulfill their parity checks. There are techniques in the literature that uses soft-values of the target VNs as convergence criteria. However, we are interested in hard-value based parity checks, because the parity-checks are already available.

A new criterion called PSC which is based on reliable VNs was proposed in [8]. The authors distinguish the VNs inside the window into *complete VNs* and *incomplete VNs*. Complete VNs are VNs whose all the connected CNs are inside the window. This includes the target VNs as well. So, the first $W - m_s$ VNs inside the window are complete VNs. The remaining VNs in the window are the incomplete VNs. This is illustrated in Figure 4.1. The green (vertically hatched) region indicates the complete VNs and brown (horizontally hatched) region indicates the incomplete VNs. The blue (hatched) region indicates the

complete CNs and the red (back hatched) region indicates the target CNs. The authors say that the complete VNs are more reliable than the incomplete ones as they get more updates from VNs from left of the window which are more reliable that are to the right. Hence, the parity checks of only the complete CNs, i.e., the CNs connected to only the complete VNs are considered as convergence criterion.

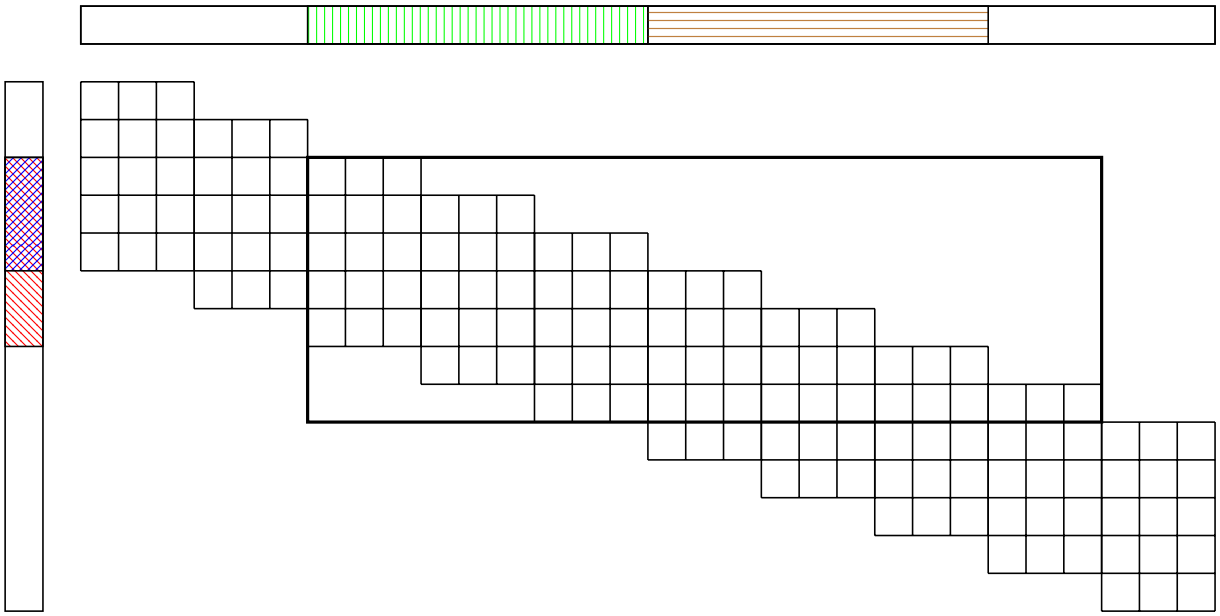


Figure 4.1: PCM illustrating complete-VNs, incomplete-VNs and complete-CNs.

4.2 Base Decoder Configuration

Before discussing the details of the techniques that are developed in this work, it is essential to define a decoder configuration on top of which the techniques are intended to apply. This allows better understanding of the techniques and evaluation of the simulation results. Let us call the decoder without our improvements as Base Decoder (BD). The BD uses the sliding-window technique to decoder the BPL codes. The window slides from the left end of the PCM to the right end. Within each window, serial scheduling is performed by updating the CNs from top to bottom for a maximum of $I_{\mathrm{BD}} = I$ iterations.

In the last window position, i.e., when the window touches the right most column of the PCM, all the VNs inside the window are considered as target VNs. Hence, the convergence criterion is to check if all CNs inside the last window fulfill their parity checks. This idea was proposed in [21] as Early-Stopping rule. We adapt this technique in our BD to minimize the total number to iterations. The convergence criterion for all the window instances in our BD is that the target CNs should fulfill their parity checks.

We now know that the BPL codes are terminated using the zero-tail termination. The zero-tail bits and their positions are always known at the receiver. Hence, the LLRs of the zero-tail bits are made to $+\infty$ which indicate that the value of these bits are most certainly 0.

The configurations of the BD is summarized below.

1. BP based windowed decoder and the window moves from left to right.
2. Maximum number of iterations for each window is $I_{\mathrm{BD}} = I$.
3. Serial scheduling with an update order of CNs from top to bottom.
4. In the last window, all VNs are considered as target VNs.
5. Target CNs are considered for convergence criterion.

4.3 Left-to-Right-to-Left Decoder

The Zigzag decoder performs better than the sliding-window decoder but the decoding complexity in terms of number of iterations is higher. The implementation complexity is also high due to the nature of the decoder. Also, the Zigzag decoder uses flooding scheduling in its windows which is subpar compared to serial scheduling. Hence, the aforementioned factors motivates us to develop a decoder that moves in both directions in the codeword but has similar complexity and better performance than a sliding-window decoder.

In this thesis, we propose a *Left-to-Right-to-Left (LRL) decoder*. The LRL decoder moves the window from left to right and from right to left of the PCM unlike the BD that moves the window only from left to right. As already mentioned earlier, the bits in the left and right of the codeword are more reliable than the bits in the middle due to the lower CN degrees in both the ends of the PCM. This characteristic of the codeword can be seen in Figure 4.2 which plots the probability of error for all the bits in the codeword. We can see that the bits in the left and right of the codeword have lower probability of error than the bits in the middle. During the first decoding phase of the LRL decoder, the window moves from the first window position to the last window position of the PCM. During the second phase, i.e., after decoding the last window position, the window moves to the left till it reaches the first window position. Figure 4.3 illustrates the LRL decoder.

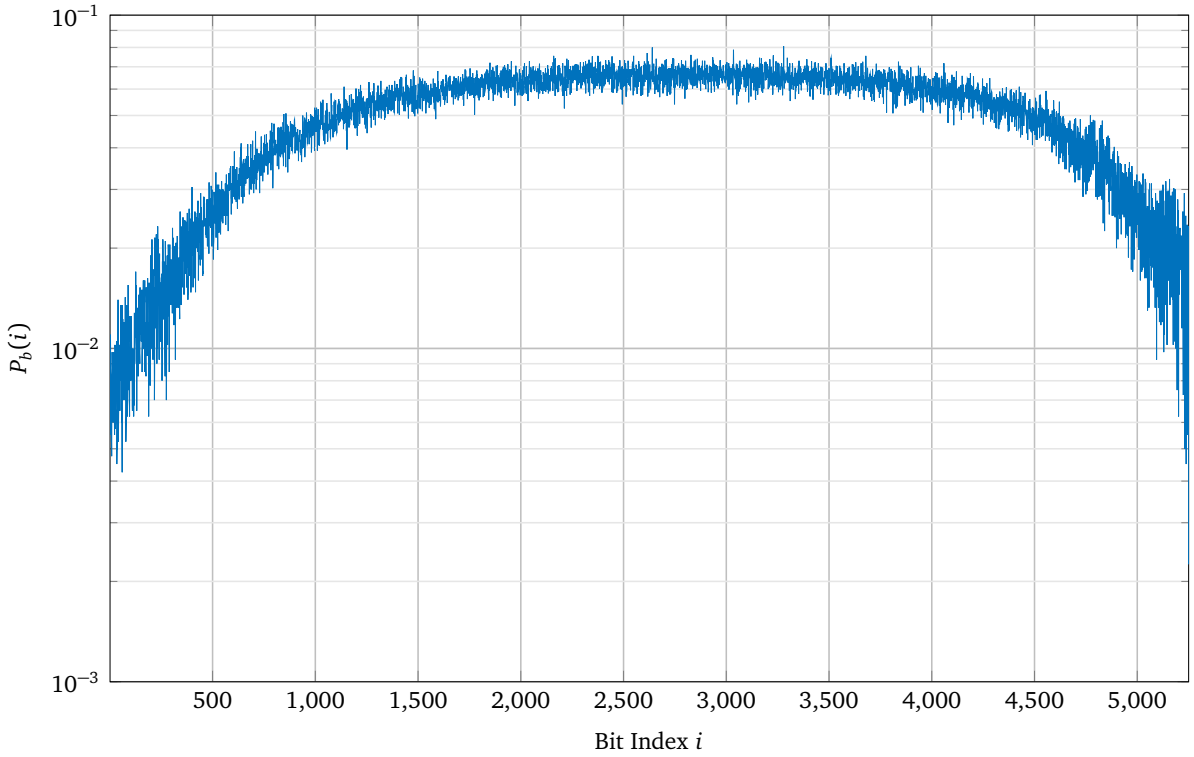


Figure 4.2: $P_b(i)$ for each bit in the codeword of a code with $R_\infty = 2/3$, $n_m = 3500$ at $\zeta = 2.8$ dB. Termination bits are excluded.

During the first decoding phase, the information from the VNs in the left propagates to the VNs to the right as the window moves forward. Then during the second decoding phase, the information from the VNs in the right propagates to the VNs to the left as the window moves backward. Thus, the highly reliable information from both ends of the codeword flows to the other parts of the codeword. The maximum number of iterations within each window is half of the maximum number of iterations in the BD, i.e., $I_{\text{LRL}} = I_{\text{BD}}/2$. This is done to maintain the same decoding complexity as the BD because each window decoding in the LRL decoder is performed twice.

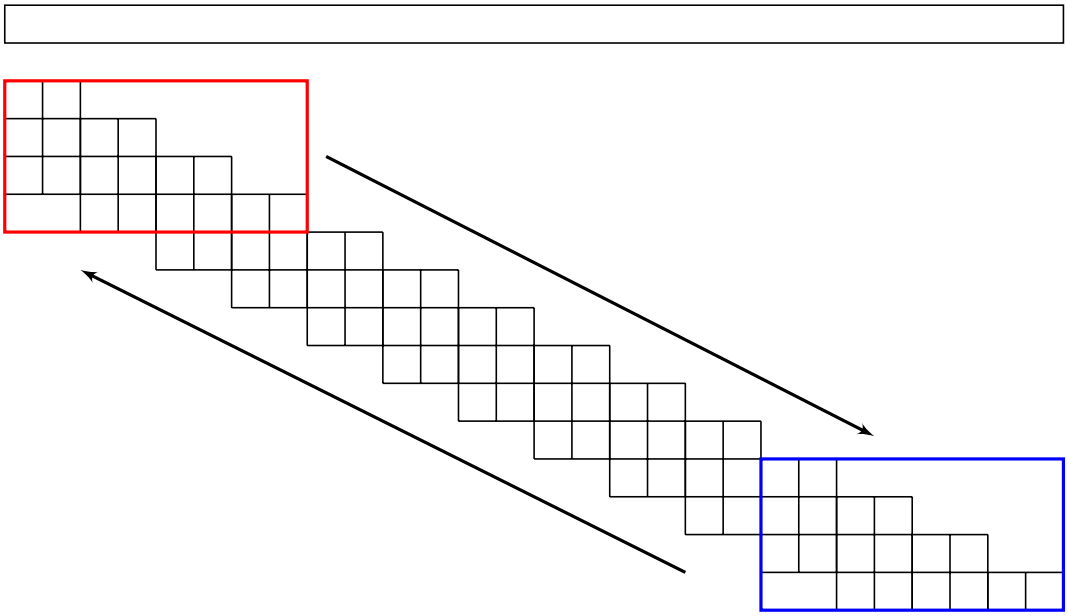


Figure 4.3: PCM illustrating LRL decoder.

Within the LRL decoder, we propose two different window configurations. They are illustrated in Figure 4.4. The red box indicates the window. The illustration in the left shows a window configuration where the left most VNs (blue hatched) are the target VNs. The order of CN update is from top to bottom as indicated by the brown arrow. Let us call this window configuration-I. The illustration to the right shows that the right most VNs in the window are the target VNs. The order of CN update is from bottom to top as indicated by the brown arrow. Let us call this window configuration-II.

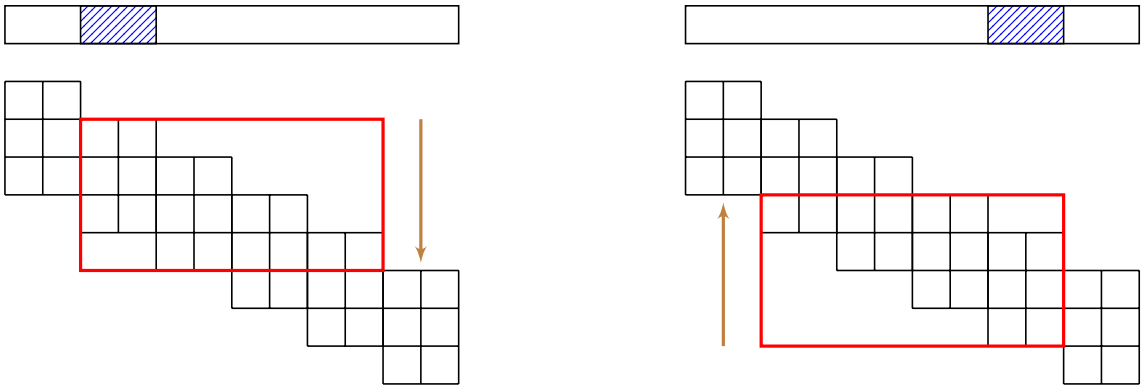


Figure 4.4: Different window configurations used in LRL decoder. Left image illustrates LRL window configuration-I and right image illustrates LRL window configuration-II.

We discuss the performance of the LRL decoder with both window configurations. At first, we use LRL decoder with window configuration-I in both phases of the LRL decoder. Secondly, we use the window configuration-I during the first phase and window configuration-II during the second phase. During the second phase, the CNs are updated from bottom to top so that the information from right of the codeword flows to the left more consistently as the window moves backward. The simulation results of both the configurations are evaluated in Chapter 5.

4.4 Improved Partial-Syndrome-Check

Although the PSC rule proposed in [8] reduces the number of iterations with same performance, they are suitable only for small windows where $W \leq 2m_s + 1$. For larger windows, the number complete VNs are greater than the number of incomplete VNs which makes the number of complete CNs to be greater than the number of target CNs, i.e., the CNs connected to target VNs. So, for larger windows the PSC rule actually increases the number of iterations. These factors motivates us develop a convergence criterion that depends on the window size.

Thus, we propose a IPSC rule. The PSC uses only complete CN as the convergence criterion. In the IPSC rule, we introduce an additional convergence criterion that involves target CNs. The number of target CNs is always the same for a code. It is given by

$$\Upsilon_t = m_s + 1. \tag{4.1}$$

The number of complete CNs is dependent on W and is given by

$$\Upsilon_c = W - m_s. \tag{4.2}$$

The maximum W for which the number of complete CNs is less than or equal to the number of target CNs is given by

$$\Upsilon_c \leq \Upsilon_t \tag{4.3}$$
$$W - m_s \leq m_s + 1 \tag{4.4}$$
$$W \leq 2m_s + 1. \tag{4.5}$$

When the window size $W > 2m_s + 1$ the target CNs are considered for convergence criterion because the number of complete CNs are greater than the number of target CNs.

Table 4.1 summarizes the IPSC technique. The simulation results of the IPSC are evaluated in Chapter 5.

Window Size	**Convergence Criterion**
$W \leq 2m_s + 1$	Complete CNs.
$W > 2m_s + 1$	Target CNs.

Table 4.1: Convergence criteria for IPSC

Figure 4.5 shows Υ_c and Υ_t for BPL code with $R_\infty = 2/3$ and various window sizes. We see that after the point where $W = 2m_s + 1$, the $\Upsilon_t < \Upsilon_c$.

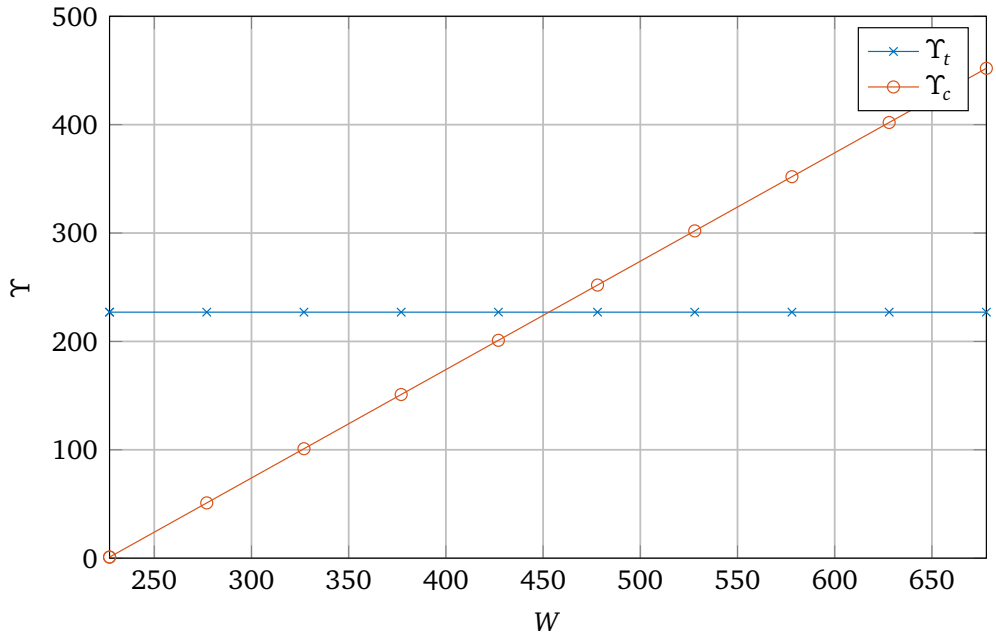


Figure 4.5: Number of complete and target CNs for code with $R_\infty = 2/3$ and $m_s = 226$.

5 Simulation Results and Evaluation

In this chapter, we analyze the simulation results of our proposed techniques. We start by discussing the simulation setup. Then, we analyze the plots to evaluate the performance of our techniques.

5.1 Experiment Setup

For our simulations, we used the baseband system model described in Section 2.4. Table 5.1 lists the different parameters of the simulation setup. All the simulations are performed with these parameters unless otherwise specified.

Parameter	**Symbol**	**Value**
Number of information bits	n_m	3500
Codeword Length	n_L	6390
Actual code rate	R_L	0.6216
Asymptotic code rate	R_∞	0.6667
Number of termination bits	m_t	380
Modulation	o	QPSK
Window size	W	300
Maximal Number of iterations	I	10
Number of runs	Γ	4000

Table 5.1: Experimental settings for simulations.

In most of our results we plot the Block-Error Rate (BLER) P_L instead of Bit-Error Rate (BER) P_b because in modern packet based transmission system, the whole block is discarded in physical layer if they are incorrect. Hence, it is better to evaluate the performance by BLER than BER. We plot Total Number of Edge Updates (TNEU) to measure the decoding complexity. All the simulations results are averaged over Γ runs.

5.2 Effect of Zero-tail Termination

In Chapter 3, we concluded that the termination for BPL codes are performed through zero-tail termination. Here, we evaluate the effect of zero-tail termination on the probability of error of each bit in the codeword. Figure 5.1 shows the $P_b(i)$ of each bit in the codeword when the zero-tail bits are not known at the receiver. Figure 5.2 shows the $P_b(i)$ of each bit in the codeword when zero-tail bits are known at the receiver, i.e., at the decoder the LLRs of zero-tail bits are saturated to $+\infty$. The probabilities are calculated using (2.50). The plots include the information bits and the parity bits but do not include the termination sequence.

From Figure 5.2 we can see that when zero-tail bits are know at the receiver, the decoder reduces the $P_b(i)$ of the information and parity bits in the right of the codeword. Only the rightmost part of the

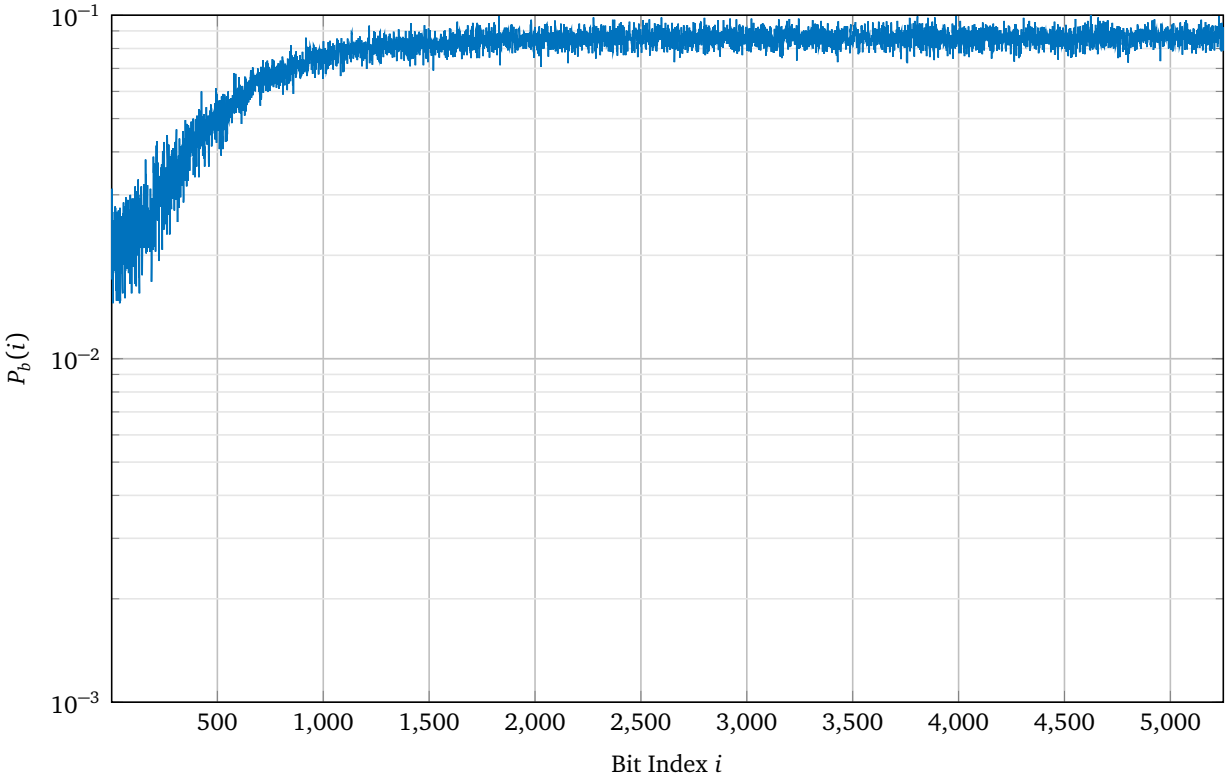


Figure 5.1: Probability of error for information and parity bits in the codeword at $P_b = 5 \cdot 10^{-2}$. Zero-tail bits are not known at the receiver.

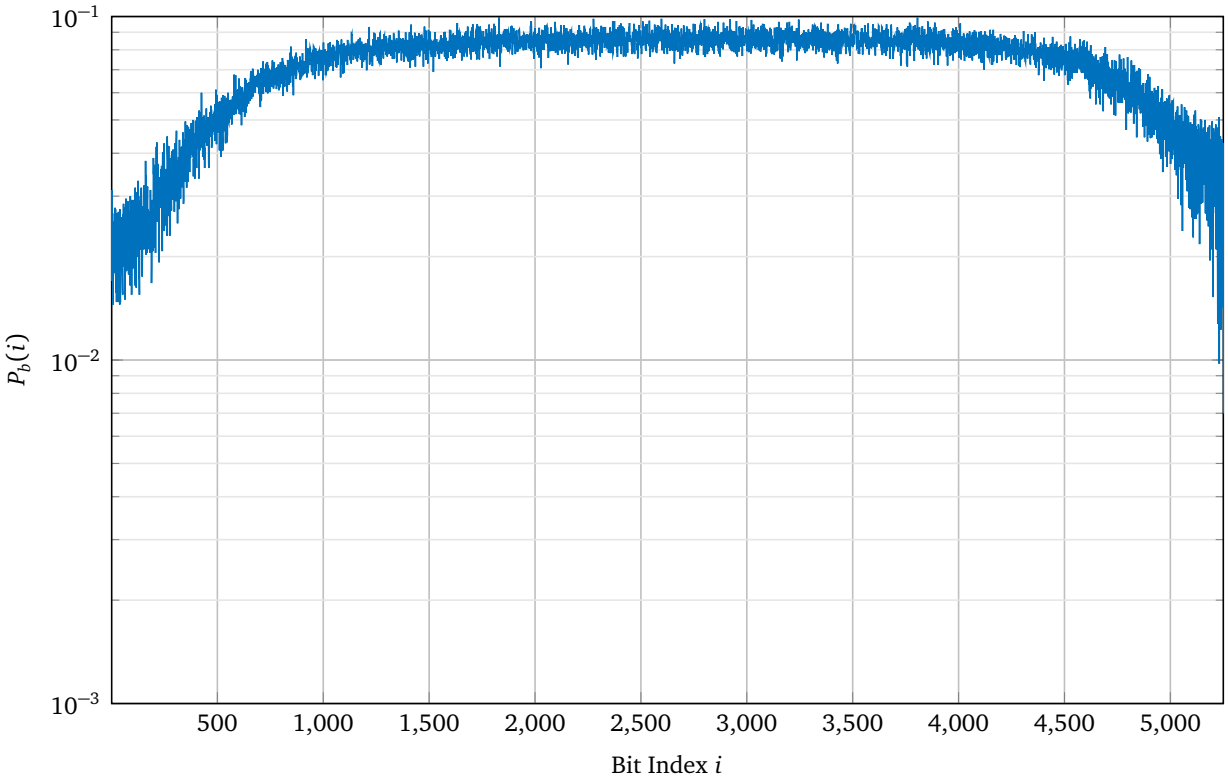


Figure 5.2: Probability of error for information and parity bits in the codeword for $P_b = 5 \cdot 10^{-2}$. Zero-tail bits are known at the receiver.

codeword is affected because of the limited window size. With full-block decoder, the effect of having known bits would also improve the leftmost bits of the decoder. On the other hand, Figure 5.1 shows that the lack of knowledge of zero-tail bits at the receiver does not reduce $P_b(i)$ at the end of the codeword. Thus, the knowledge of zero-tail bits at the decoder effectively reduces the CN degrees at the right of the PCM but not as low as in a properly terminated codeword. Hence, zero-tail termination is an acceptable alternative to proper termination.

5.3 Evaluation of Base Decoder

Here, we analyze the decoding performance and complexity of our BD over different window sizes. Figure 5.3 shows the overall BLER P_L for different window sizes. Figure 5.4 shows the TNEU.

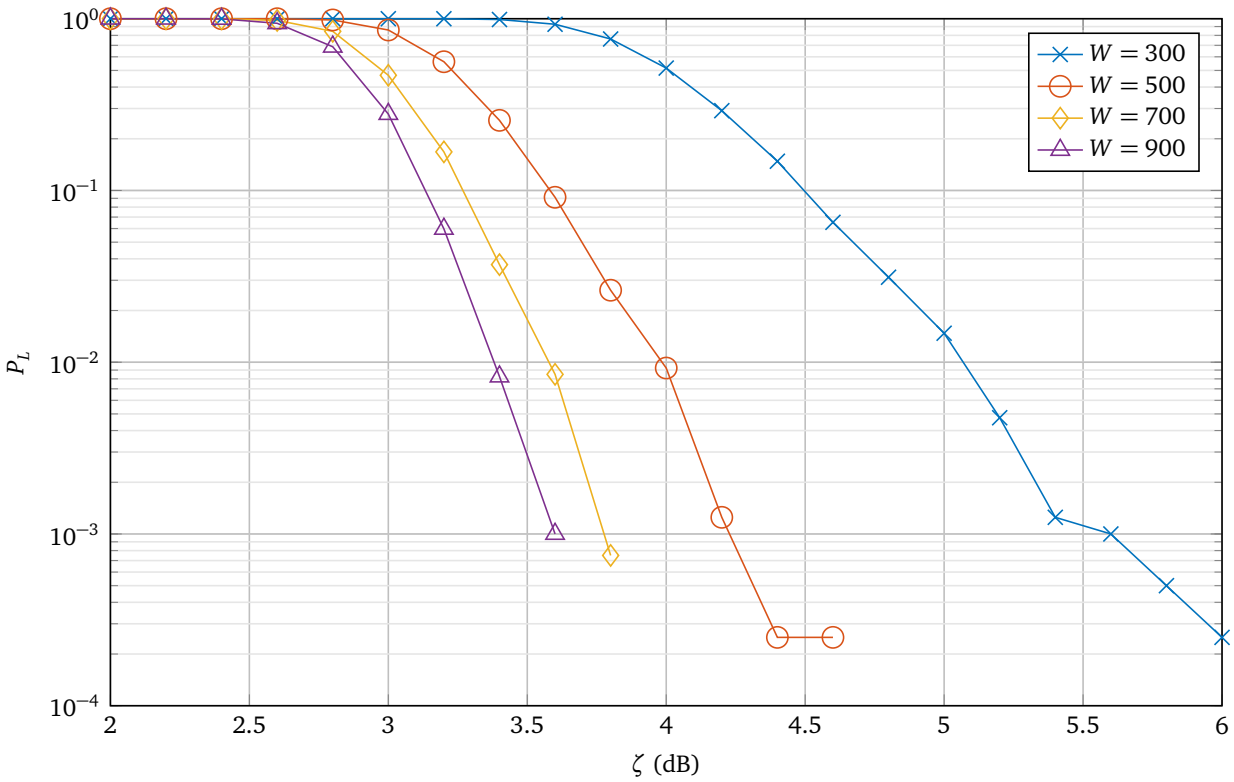


Figure 5.3: P_L vs ζ of the Base Decoder (BD).

In Figure 5.3, we see that the BLER improves with increasing window size W. When the window size is increased, more edges are included in the window enabling information to flow between a larger number of VNs. At $P_L = 10^{-1}$, the distance between $W = 300$ and $W = 900$ is about 1.4 dB. Figure 5.4 shows that the TNEU is high for larger window in the low-SNR region because the windows do not converge and thereby exhausts all iterations. In the waterfall region, i.e., the SNR domain where the BLER continuously decreases, larger windows converge quicker. For example at $\zeta = 3.5$ dB, there is a 75% decrease in TNEU. And in the error floor region, i.e., the SNR domain where there is no further decrease in BLER, the TNEU for all W is about the same.

We also plot the performance over different code rates. Figure 5.5 shows different BLER plots for all available rates R_∞ for BPL codes. It is well known that codes with lower rate requires lower SNR to achieve the same BLER as the code with higher rates.

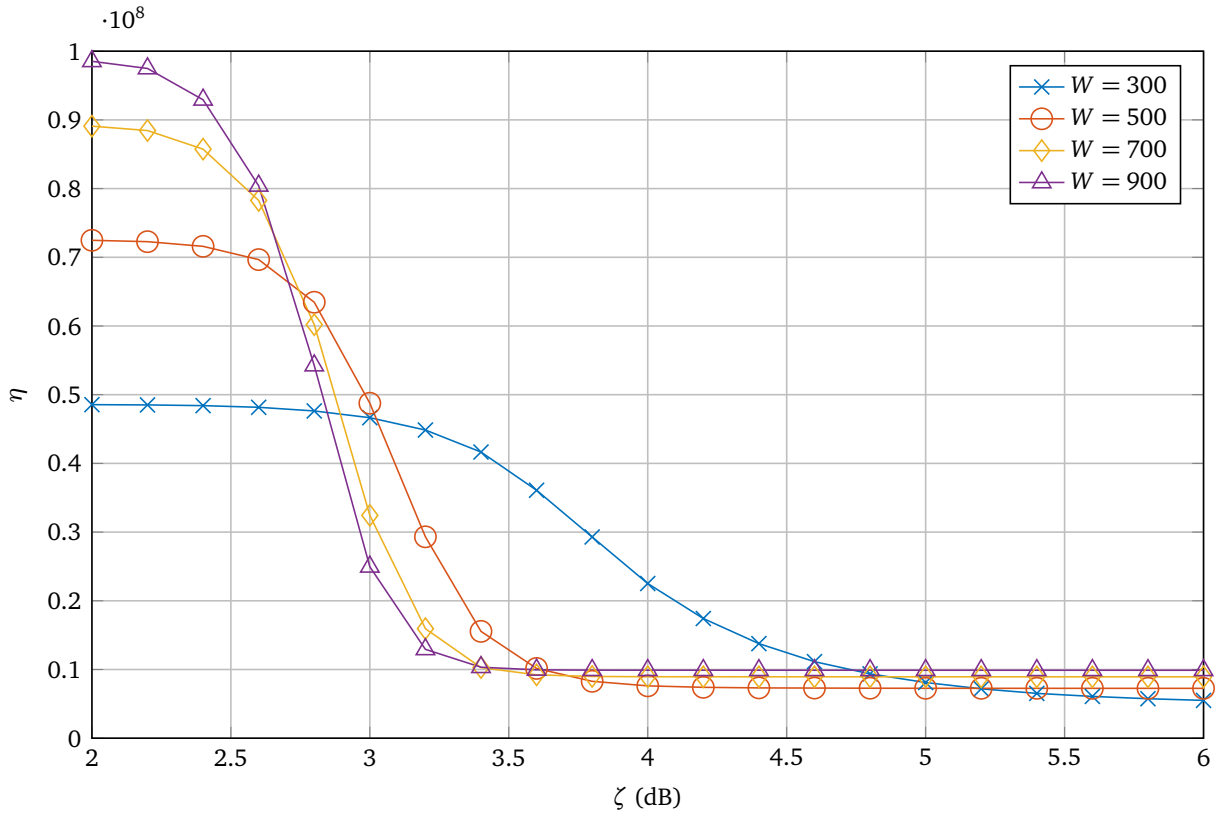


Figure 5.4: η vs ζ of the Base Decoder (BD).

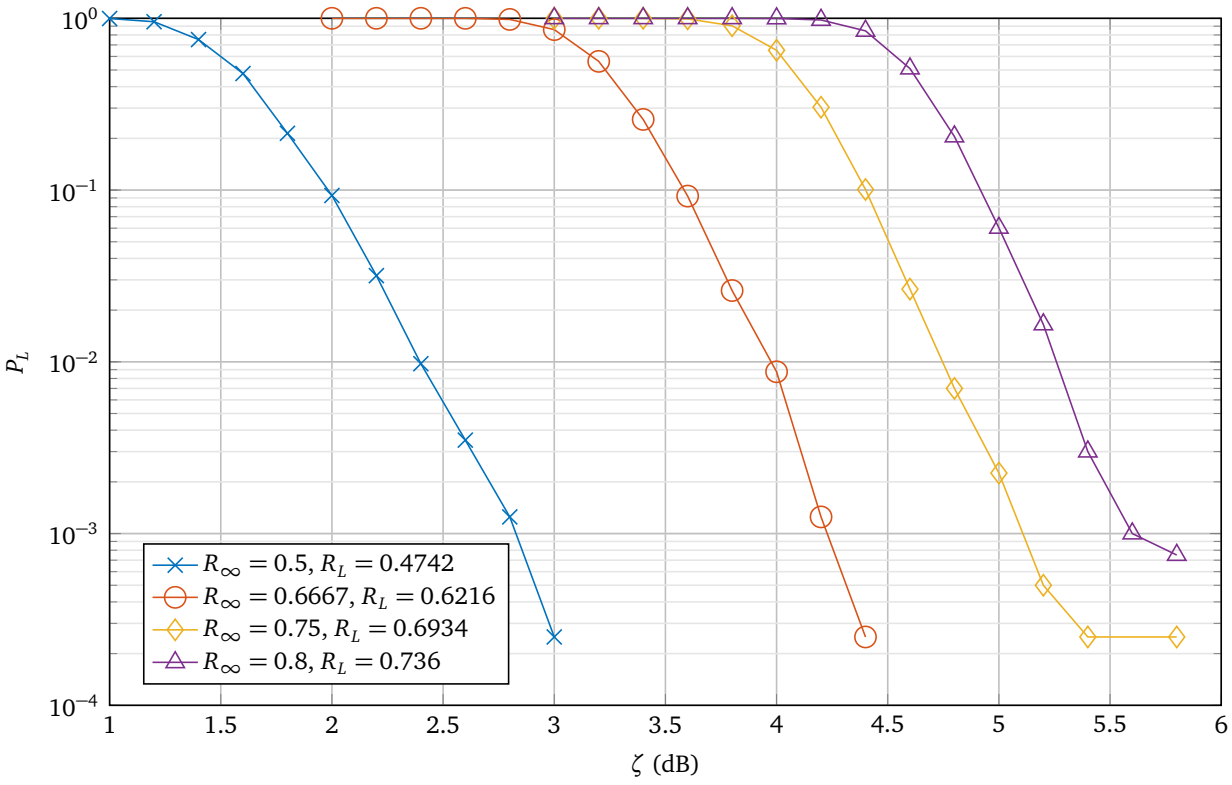


Figure 5.5: P_L vs ζ of the Base Decoder (BD) for all available R_∞ with $W = 500$.

5.4 Evaluation of Left-to-Right-to-Left Decoder

Now, we compare the performance of LRL decoder configuration-I, LRL decoder configuration-II with the BD. Figure 5.6 shows the P_L of BD and LRL decoders with window size $W = 300$. Similarly, Figure 5.7 shows the η of BD and LRL decoders with window size of $W = 300$. Table 5.2 shows different window configurations used in simulation.

Simulation Configuration	First Phase	Second Phase
LRL configuration-I	Window configuration-I	Window configuration-I
LRL configuration-II	Window configuration-I	Window configuration-II

Table 5.2: LRL decoder with different window configurations.

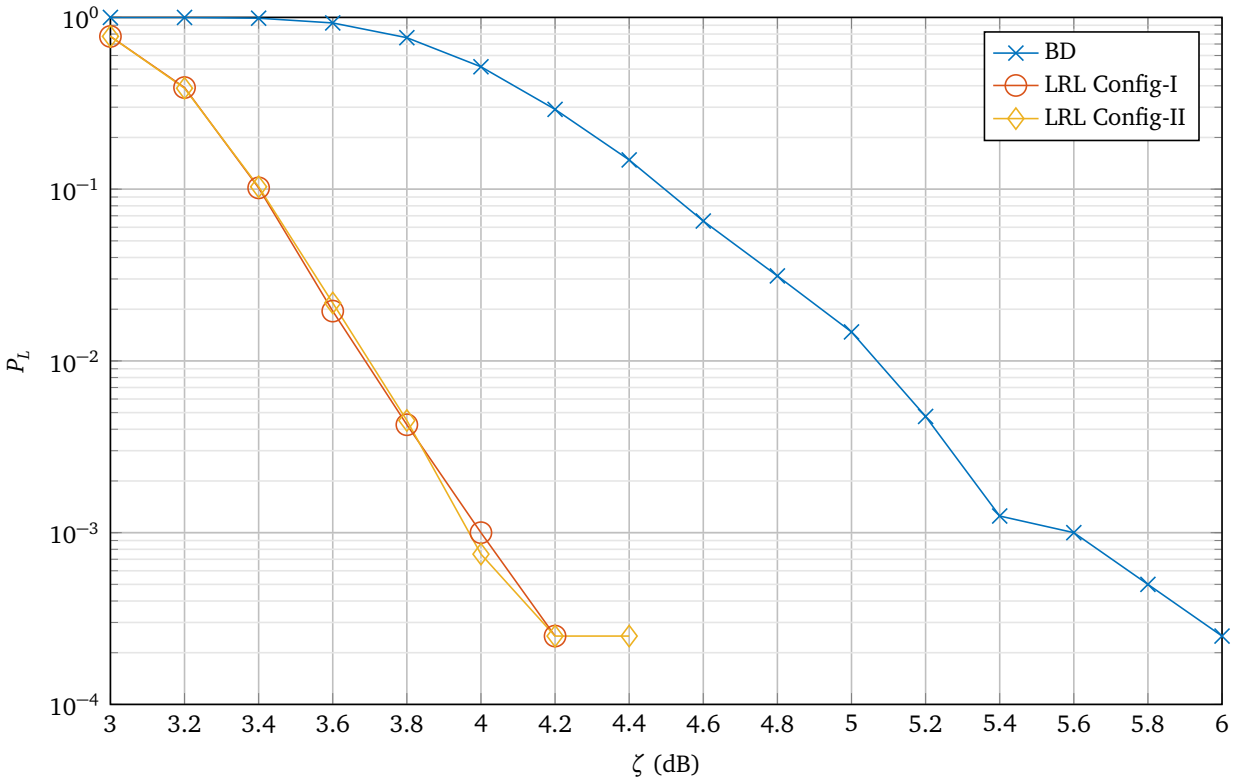


Figure 5.6: Comparison of P_L between the Base Decoder and LRL decoder.

In Figure 5.6, we see a significant decrease in P_L of the LRL decoder in both configurations compared to the BD. Near the error-floor region where $P_L = 1.5 \cdot 10^{-4}$, there is a ζ gain of about 1.6 dB. In the beginning of waterfall region where $P_L = 3 \cdot 10^{-1}$, there is a ζ gain of about 0.9 dB. The gain in individual BER $P_b(i)$ is seen in Figure 5.8. We can see that just before the end-termination of the codeword, i.e., between 3500-th and 4000-th bit, there is a $P_b(i)$ difference of $2 \cdot 10^{-2}$. And in the rest of the codeword, there is a $P_b(i)$ difference of only $1 \cdot 10^{-2}$. It indicates that the second phase of the LRL decoder has improved the certainty of the bits through the information from the bits in the end-termination of the codeword. Hence, the proposed LRL decoder is better than the BD in terms of decoding performance.

In Figure 5.7, we see that the overall complexity of both LRL are less than the complexity of BD. In the waterfall region, i.e., starting from $\zeta = 3$ dB there is a 20 to 40% decrease in η for both LRL decoder configurations. The decrease is mainly because the second phase of the decoder propagates

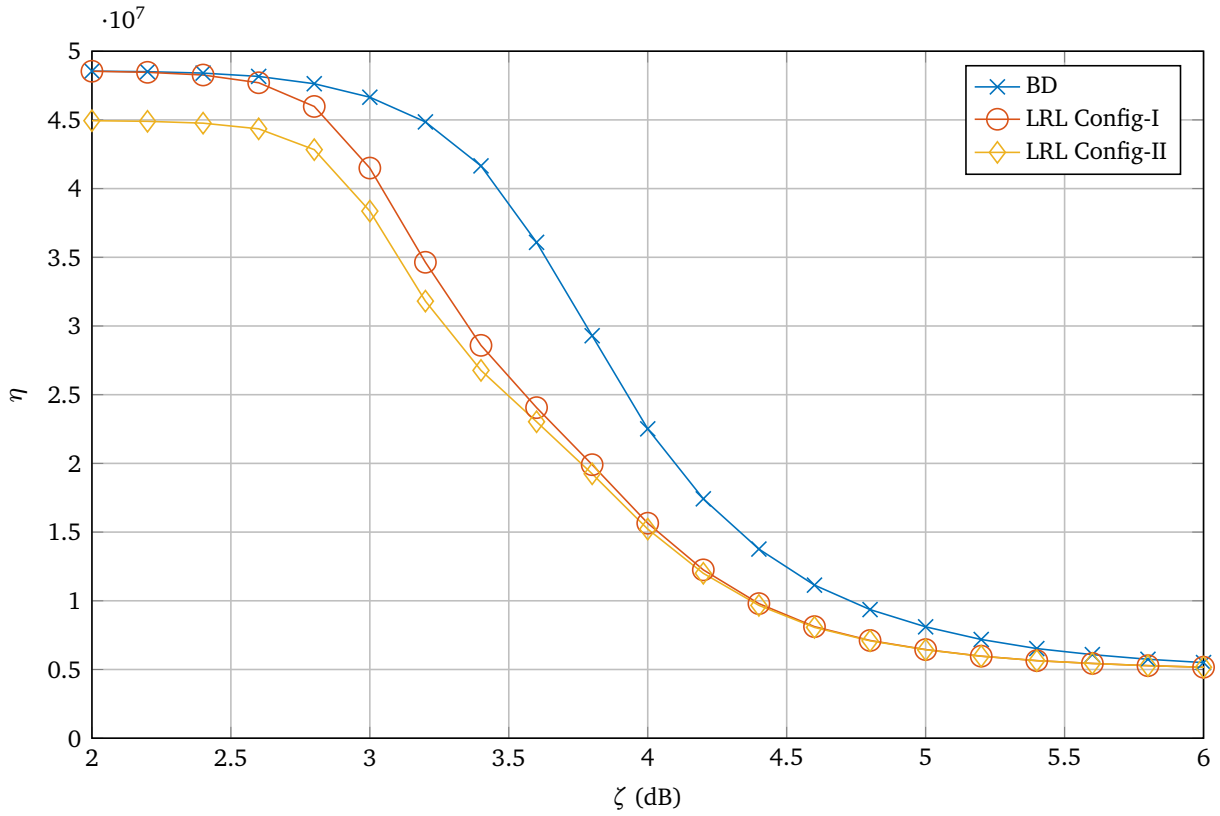


Figure 5.7: Comparison of η between the Base Decoder and LRL decoder.

the reliable information from the end-termination to rest of the codeword, increasing the convergence speed of the windows. At low-SNR and waterfall regions, we can notice that the LRL configuration-II decoder converges faster than the configuration-I decoder. This faster convergence speed is because the target VN are placed at the right side of the window and the CNs are processed from bottom to top. On continuing through the waterfall region, the gap between BD and LRL decoder closes. At the error-floor region, the complexity of all three decoders are the same. At error-floor region, the SNR is so high that the BD's windows converge as quick as the LRL decoder's windows.

Figure 5.9 and Figure 5.10 compares the performance and complexity between the BD and both LRL decoder configurations for window size $W = 700$. We see that at the waterfall region where $P_L = 9{\cdot}10^{-2}$, the ζ gain is about 0.3 db. This performance gain is very less compared to the gain obtained with $W = 300$. The decrease in performance gain is because as the window size increases the BP's performance is better as seen in Figure 5.3. Thus, further increasing the window size, increases the performance of both BD and LRL decoders by different steps and finally approaching the performance of a full-block decoder, i.e., a decoder with maximum W.

We see that as W increases, there is a slight decrease in performance of LRL configuration-II decoder compared to configuration-I. This difference in performance is because of the less window positions at the left side of the PCM for LRL configuration-II. One could argue that the same factor would affect the performance of LRL configuration-I at the right side of the PCM thus, on average yielding the same performance. But from Section 3.2, we know that the start-termination has lower CN degree than the zero-tailed end-termination. Hence, the more number of window positions allowed by the LRL configuration-I decreases the BLER compared to LRL configuration-II.

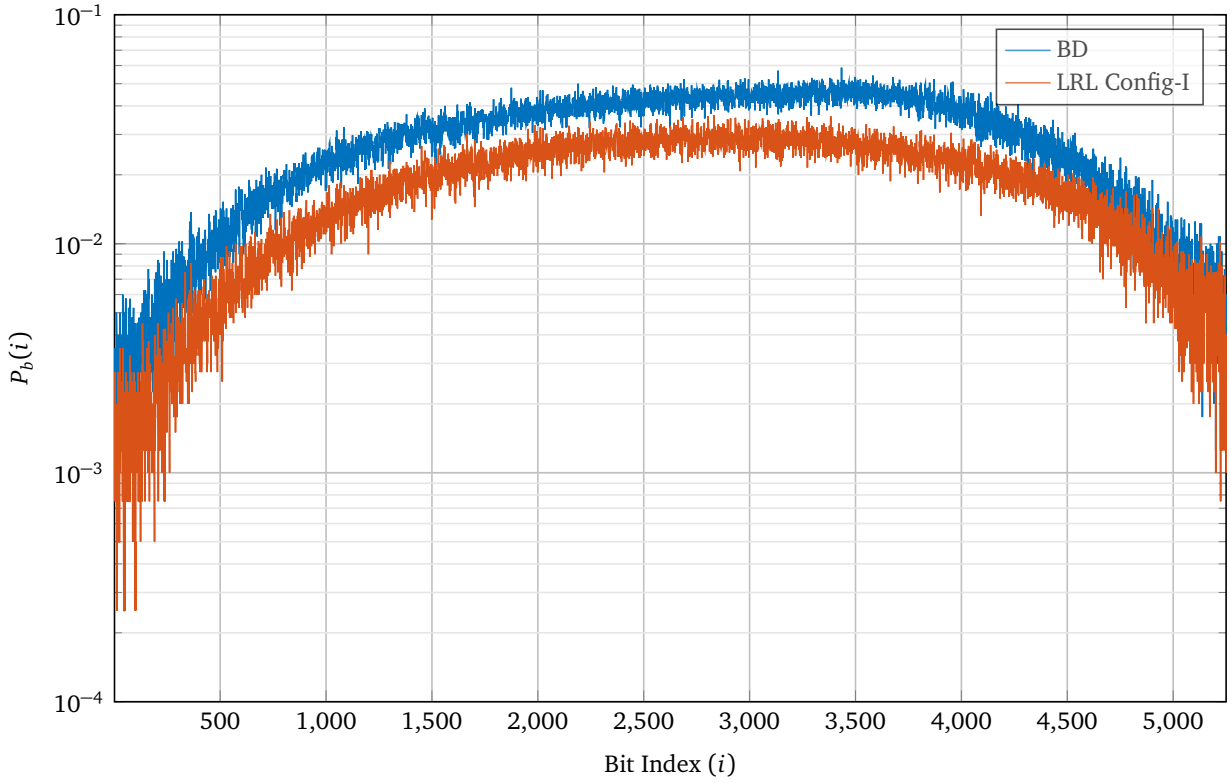


Figure 5.8: $P_b(i)$ for BD and LRL decoder for $W = 500$ and $P_b = 5 \cdot 10^{-2}$.

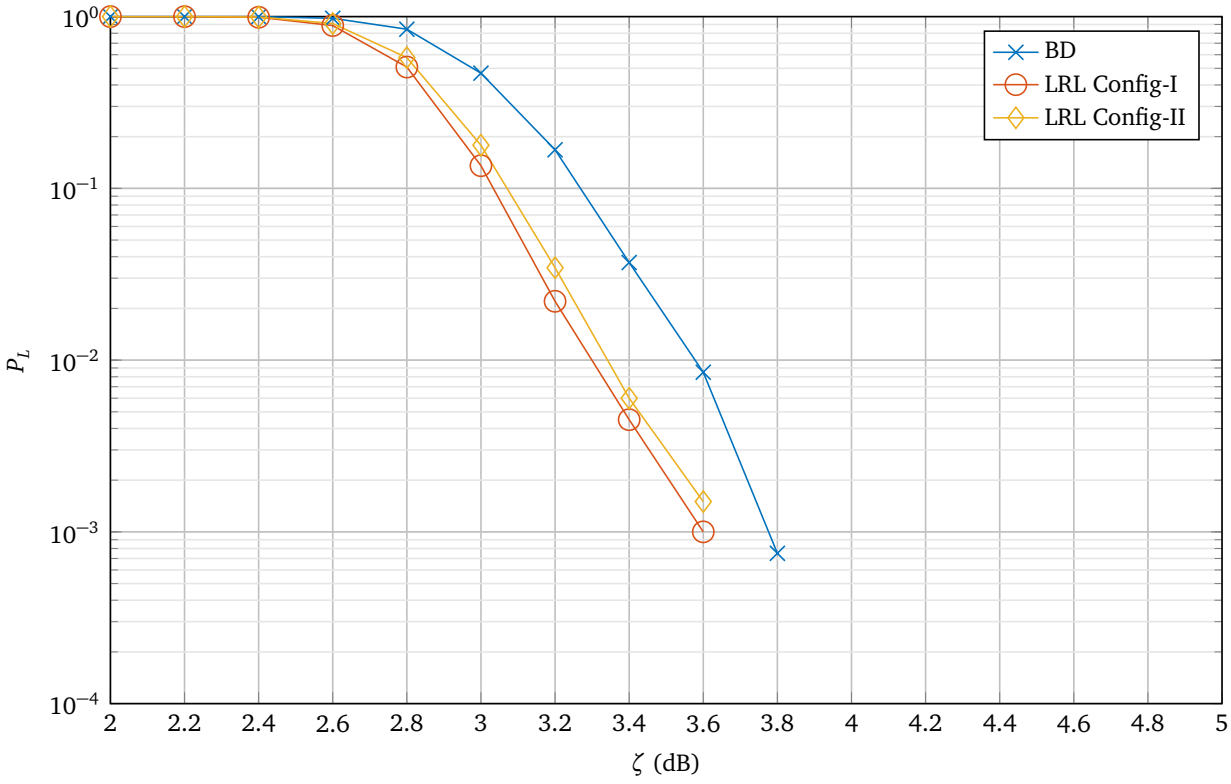


Figure 5.9: Comparison of P_L between the BD and LRL decoder with $W = 700$.

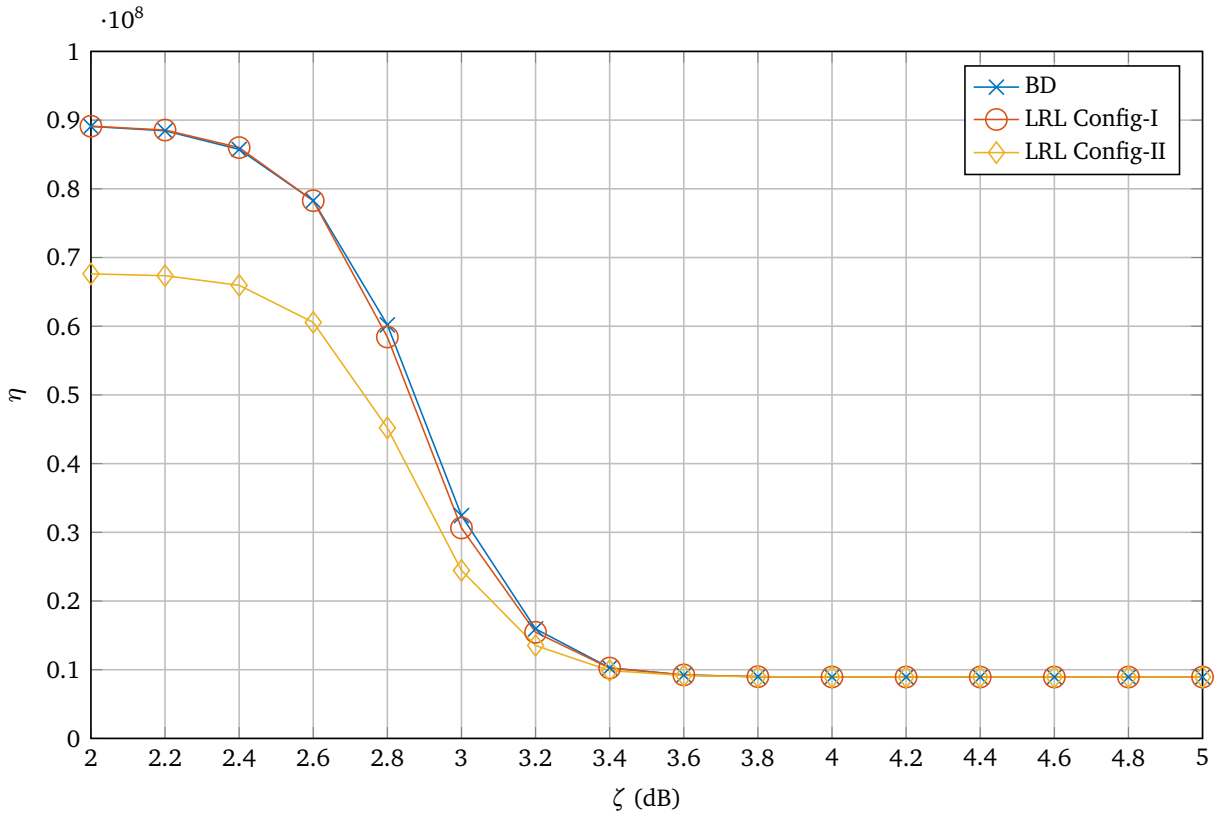


Figure 5.10: Comparison of η between the Base Decoder and LRL decoder with $W = 700$.

5.5 Evaluation of Improved Partial-Syndrome-Check Technique

Here, we evaluate the performance of our IPSC technique. Figure 5.11 shows the decoding performance of different convergence criteria with $W = 300$ and $W = 700$. Figure 5.12 shows the complexity of different convergence criteria with $W = 300$ and $W = 700$.

In Figure 5.11, we see that there is no noticeable difference in P_L when complete CNs or target CNs are considered for convergence criterion. However, Figure 5.12 shows a difference in η between both convergence criteria. For $W = 300$, i.e., $W \leq 2m_s + 1$, there is a decrease in η in the waterfall region when complete CNs are considered for convergence criteria as promised by PSC rule.

Now, for $W = 700$, ie., $W > 2m_s + 1$, there is a decrease in η in the waterfall region by up to 34% when target CNs are considered for convergence criteria. This decrease in decoding complexity with the same performance tells us that the windows converge faster for $W > 2m_s + 1$ and hence, its sufficient to check the parity-checks of only the target CNs.

Table 5.3 summarizes the results from our evaluations. The results are in comparison with our BD.

Technique	Performance Gain	Complexity Reduction
LRL Decoder	0.9 – 1.6 dB	$< 40\%$
IPSC Rule	–	$< 34\%$

Table 5.3: Summary of results of our techniques compared to the BD.

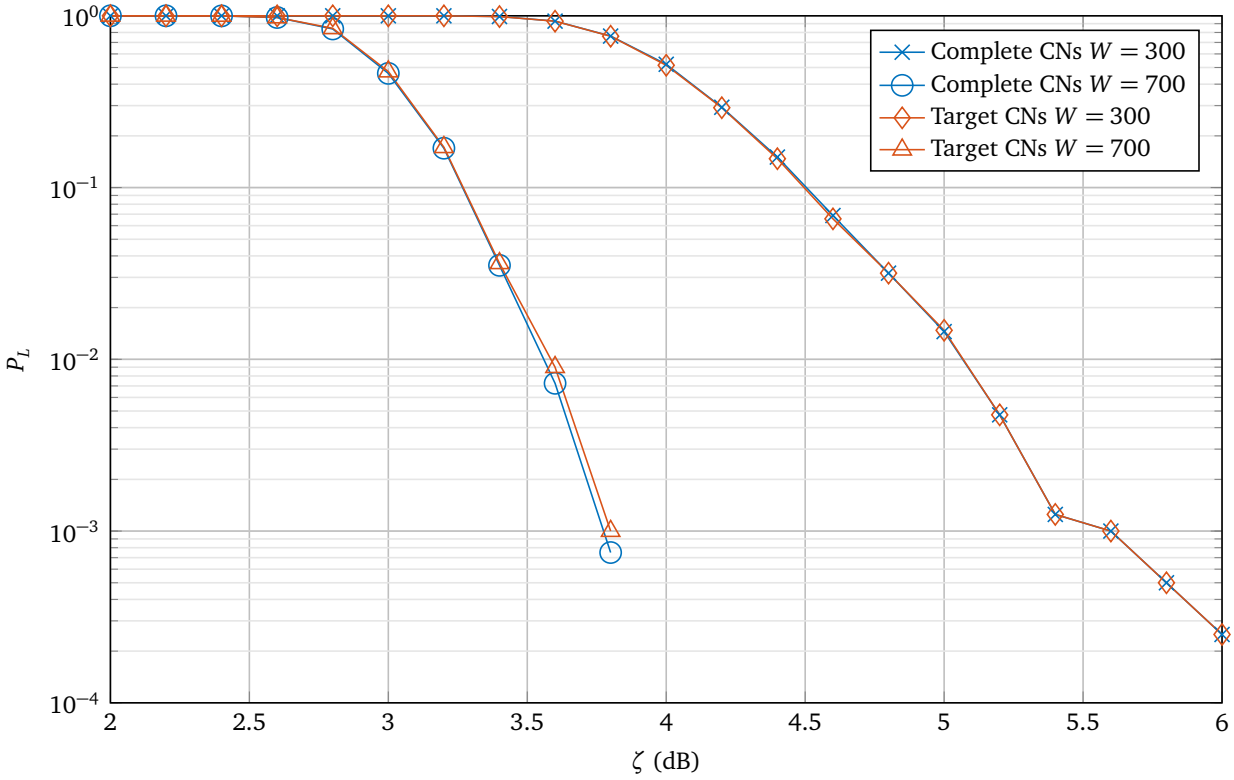


Figure 5.11: Comparison of P_L between different convergence criteria for $W = 300$ and $W = 700$.

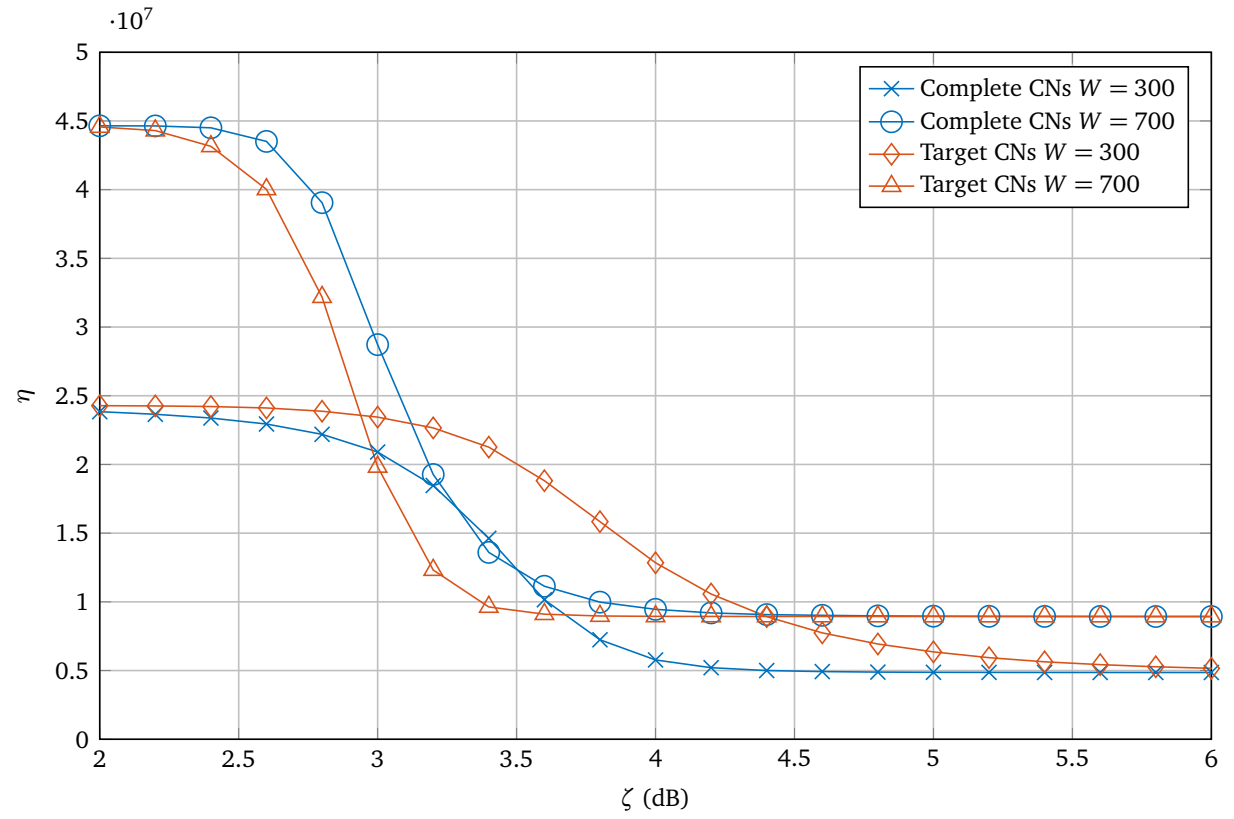


Figure 5.12: Comparison of η between different convergence criteria for $W = 300$ and $W = 700$.

6 Implementation Aspects

In this chapter, we discuss how the encoder and decoder are implemented. We start with describing the encoder's implementation and reasons for the chosen method. Then we discuss the implementation of the decoder and the implications of different implementations on the execution complexity.

6.1 Variable Node Storage Memory Format

The hard values of the VNs or the bits in the codeword can be stored in two different ways:

1. Byte variable for a bit: In x86 architecture, the smallest addressable memory unit is a byte. Each byte of memory contains eight bits out of which the Least Significant Bit (LSB) represents one VN. An example of such a storage scheme is shown in Figure 6.1. This form of storage allows us to directly access each bit as `uint8` and use it to perform signal processing as shown in Figure 6.2.

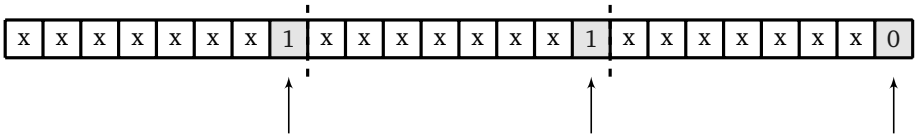


Figure 6.1: Three bits of value 0, 1 and 1 are stored in single byte each. The arrow indicates the position of LSB where the bit is stored in each byte. The bits marked with x are unused. Note that the bytes are represented in little-endian format.

Figure 6.2: Signal processing blocks directly access and use use the bits.

2. Packed byte of bits: Each byte of memory contains eight bits representing eight VNs. An example of such storage is shown in Figure 6.3. With this form of storage, additional functions are required to access and store each bit from and to its corresponding position because the minimum quantity of bits that can be accessed from the memory at once is a byte or `char` or `uint8`. This is shown in figure 6.4.

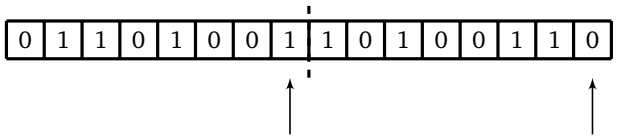


Figure 6.3: Packed bits of bytes.

In our implementations, we use the packed-byte-of-bits format. This format reduces the memory requirements for storing information and codeword bits by a factor of eight. The algorithms for *Load Bit*

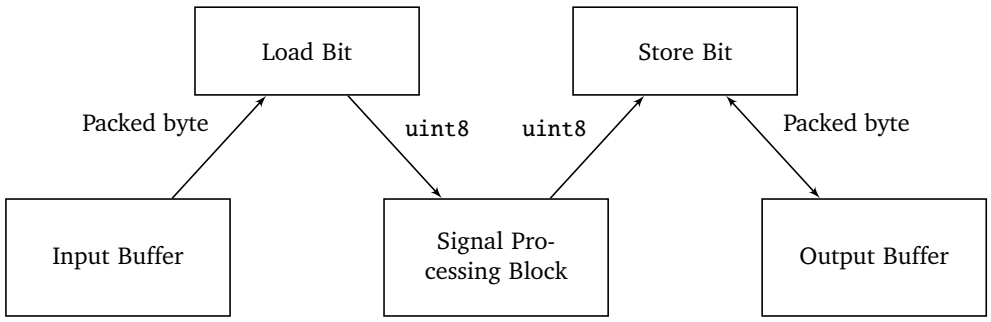


Figure 6.4: Access and use of bits using helper functions. The double arrow indicates that the Store Bit function reads and writes to the output memory.

Storage Format	Memory Required (Bytes)
Byte for a bit	n_m
Packed byte of bits	$\lceil \frac{n_m}{8} \rceil$

Table 6.1: Memory required to store hard values of VNs.

an *Store Bit* are shown in Algorithm 1 and 2, respectively. Table 6.1 shows the memory required to store the hard values of VNs.

Data: j (Bit index in buffer)
Result: v (Corresponding bit value as `uint8`)
i (Calculate byte index from j);
b (Calculate bit index within the byte);
v (Read the i-th byte from buffer);
Left shift v by $(8-b)$ bits;
Right shift v by 7;

Algorithm 1: Load Bit

Data: j (Bit index in buffer), c (Bit value as `uint8`)
Result: c stored in the right bot position of the buffer
i (Calculate byte index from j);
b (Calculate bit index within the byte);
v (Read the i-th byte from buffer);
if *c equal to 1* **then**
  v bitwise OR with 1 left shifted by b bits;
else
  z (NOT of 1 left shifted by b bits);
  v bitwise AND with z;
end
Store the resulting byte in i-th byte position in buffer;

Algorithm 2: Store Bit

However, the gain in minimum memory requirement comes with a cost of increased execution complexity. Table 6.2 shows the operations performed in each call of *Load Bit* and *Store Bit*. Note that this is just one way of implementing the functions.

Table 6.3 compares the memory requirement and operations required to encode a codeword with n_m information bits.

Although the execution complexity of using Packed-byte-of-bits format is high compared to the other format, the Packed-byte-of-bits format's complexity is very negligible compared to the overall decoding

Function called	Reads	Writes	Bit Shift	Arithmetic	Logical
Load Bit	1	0	2	2	0
Store Bit	1	1	1	2	2

Table 6.2: Number of different operations performed during each call of *Load Bit* and *Store Bit*.

Storage Format	Memory Required (Bytes)	Reads	Writes	Bit Shift	Arithmetic	Logical
Byte for a bit	$3n_m$	n_m	$\frac{n_m}{n-1}$	0	0	0
Packed byte of bits	$\lceil \frac{3n_m}{8} \rceil$	$n_m + \frac{n_m}{n-1}$	$\frac{n_m}{n-1}$	$2n_m + \frac{n_m}{n-1}$	$2\left(n_m + \frac{n_m}{n-1}\right)$	$2\frac{n_m}{n-1}$

Table 6.3: Memory requirement and operations required to encode a codeword with n_m information bits and asymptotic code rate R_∞. We assume that each bits in the output buffer are accessed only once.

complexity. Hence, the Packed-byte-of-bits format is chosen in our implementation to reduce the memory requirement. Standard tools such as MATLAB^{TM} uses 64-bit double-precision floating-point storage for all variable by default.

6.2 Variable Node Indexing

The LLR values of VNs are stored in an array of memory where each element is a 32-bit floating point value. During each CN update, the BP algorithm selects the corresponding VN's LLR to perform the updates. This selection of VNs is done via indexing. There are two ways to perform the selection:

1. *Method 1*: During each CN update, compute the indices of the corresponding VNs so that the BP algorithm selects them.

2. *Method 2*: Before start of decoding, compute and store the edge indices for all CNs in the PCM.

Method 1 computes the indices on the fly during each CN update. The number of CNs depends on n_m. Thus, the complexity of calculating the indices increases linearly with I and n_m. On the other hand, method 2 does not increase the index computation complexity with increasing I or n_m. However, the memory required to store the indices of all CNs increases linearly with increasing number of CNs. Each index is stored in a `uint32`. Table 6.4 shows the memory and computation requirements for indices calculation in method 1 and method 2. We chose method 1 in our implementations as it requires less memory for the indices. The computation complexity in calculating the indices is very negligible compared to the decoding complexity.

Method	Memory Required (Bytes)	CN Index Calculations
Method 1	$12n$	$n_c I$
Method 2	$12n_c n$	n_c

Table 6.4: Memory requirement and computations required to calculate and store indices for decoding a codeword with n_m information bits and I iterations. Let the number of CNs be n_c and size of `uint32` is 4 bytes.

7 Conclusion and Outlook

The accomplishments of this work are summarized in this chapter. Some advices for future works in the windowed decoding of LDPC-CC are also given.

As mobile cellular technologies in the future adopt LDPC-CCs as error-correcting codes, there will be needs for efficient decoding algorithms. Smart phones now-a-days have multiple cameras and sensors, hence, increasing the power consumption. Also, popular applications such as 3D games and photo editors consume more power. These factors force the mobile device manufacturers to use power efficient modems. Our simulation results proved that the developed Left-to-Right-to-Left (LRL) decoder has a better Block-Error Rate (BLER) performance at a much lower decoding complexity than a conventional sliding-window decoder. The Improved Partial-Syndrome-Check (IPSC) is also proved to decrease the decoding complexity. The decrease in decoding complexity means increase in battery-power saving.

We showed that the BPL codes cannot be terminated normally due to their nature, and hence, they have to be zero-tail terminated. We also proved that the zero-tail termination effectively reduces the CN degree and hence the Bit-Error Rate (BER). Although the zero-tail termination decreases the CN degree at the end termination, it is not as low as the CN degree in the start termination. Hence, care should be taken to ensure that the codeword can be terminated in a proper manner when a code is being designed.

Several adjustments can be made to the LRL decoder. One such adjustment could be to move the window once from left to middle and right to middle of the PCM. It could arguably give better BER performance than just moving the window once from left end to right end of the PCM. Another suggestion with regard to convergence criterion is to use soft value based parity check along with the IPSC technique.

Bibliography

[1] "Access Network; GSM/EDGE Channel coding," *3GPP TS 45.003*.

[2] "Technical Specification Group Radio Access Network; High Speed Downlink Packet Access (HSDPA)," *3GPP TS 25.308*.

[3] "Technical Specification Group Radio Access Network; Requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN)," *3GPP TR 25.913*.

[4] "Technical Specification Group Radio Access Network; Study on New Radio (NR) access technology," *3GPP TS 25.308*.

[5] K. Huang, D. G. M. Mitchell, L. Wei, X. Ma, and D. J. Costello, "Performance comparison of LDPC block and spatially coupled codes over GF(q)," *IEEE Trans. Commun.*, vol. 63, no. 3, pp. 592–604, Mar. 2015.

[6] "IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications," *IEEE Std. 1901-2010*, pp. 1175–1180, 2010.

[7] "IEEE Standard for WirelessMAN-Advanced Air Interface for Broadband Wireless Access Systems," *802.16.1-2012*.

[8] P. Kang, Y. Xie, L. Yang, and J. Yuan, "Reliability-based windowed decoding for spatially coupled LDPC codes," *IEEE Communications Letters*, vol. 22, no. 7, pp. 1322–1325, Jul. 2018.

[9] M. Bossert, *Channel Coding for Telecommunications*. West Sussex, England: John Wiley & Sons, 1999.

[10] J. G. Proakis, *Digital Communication*. McGraw-Hill, 1995.

[11] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 15, pp. 533–547, 1981.

[12] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: MIT Press, 1963.

[13] A. Felström and K. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 10, pp. 2181–2191, 1999.

[14] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[15] Juntan Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Conf. Rec. 36th Asilomar Conf. Signals, Syst. Comput. 2002*, Pacific Grove, CA, USA, Nov. 2002, pp. 8–15.

[16] J. Frenzel, S. Müller-Weinfurtner, J. Huber, and R. Müller, "Static layered schedules and Core-Only parity check for the 5G new radio LDPC codes," in *12th International ITG Conference on Systems, Communications and Coding 2019 (SCC'2019)*, Rostock, Germany, Feb. 2019.

[17] C. Jones, E. Valles, M. Smith, and J. Villasenor, "Approximate-min* constraint node updating for LDPC code decoding," in *Proc. 2003 IEEE Military Commun. Conf. (MILCOM)*, Boston, MA, USA, Oct. 2003, pp. 157–162.

[18] Z. Chen, S. Bates, D. Elliott, and T. Brandon, “Efficient encoding and termination of low-density parity-check convolutional codes,” in *Proc. 2006 IEEE Global Commun. Conf. (GLOBECOM)*, University of Alberta, Edmonton, Canada, Nov. 2006.

[19] S. Abu-Surra, E. Pisek, and R. Toari, “Spatially-coupled low-density parity check codes: Zigzag-window decoding and code-family design considerations,” in *2015 Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, Feb. 2015.

[20] A. R. Iyengar, M. Papaleo, and P. H. Siegel, “Windowed decoding of protograph-based LDPC convolutional codes over erasure channels,” *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2303–2320, Apr. 2012.

[21] I. Ali, J.-H. Kim, S.-H. Kim, H. Kwak, and J.-S. No, “Improving windowed decoding of SC LDPC codes by effective decoding termination, message reuse, and amplification,” *IEEE Access*, vol. 6, pp. 9336–9346, Mar. 2018.